REFLEX CONTROL FOR ROBOT SYSTEM PRESERVATION, RELIABILITY, AND AUTONOMY

By

THOMAS S. WIKMAN

Submitted in partial fulfillment of the requirements

for the Degree of Ph. D.

Thesis Advisor: Dr. Wyatt S. Newman

Department of Electrical Engineering and Applied Physics

CASE WESTERN RESERVE UNIVERSITY

August 1994

REFLEX CONTROL FOR ROBOT SYSTEM PRESERVATION, RELIABILITY, AND AUTONOMY

Abstract

By

THOMAS S. WIKMAN

This thesis is concerned with the use of reflex-like control strategies for robot system reliability and autonomy. This thesis shows that reflex control is a useful concept for achieving robot system safety, reliability and robustness. The reflexive command filter for obstacle avoidance described in this thesis is a fast, on-line and failsafe obstacle avoidance method that protects the system from erroneous higher-level commands. The flee-reflexes introduced in this thesis are fast, on-line methods that protect the system from moving obstacles.

This thesis further shows that reflex control can be used as an important component of complex autonomous systems. Highly efficient subgoal-based

path planners utilizing the reflexive command filter as a local operator are described. The use of reflex control and fixed action patterns in an autonomous sonar-based world mapping scheme is also described. The reflexes and other low-level behaviors introduced in this thesis are functionally simple, execute very quickly, and allow for modular and incremental design.

This thesis further discusses how reflexes should be constructed to be useful, and how stability or cycling problems can be avoided when adding reflex modules to a system, or building a system consisting of reflex modules and other types of low-level behavioral modules. The objective of this thesis is to demonstrate the usefulness of reflex-like control for robot system reliability and autonomy through the numerous implementations described, and present methods for performance analysis and design of reflex-like control systems.

This thesis is dedicated to

my wife

Claudia S. Wikman

and my son

Jacob T. Wikman

Acknowledgments

There are many people that were instrumental in helping me complete this thesis and my educational goals. First, I would like to thank my thesis advisor Dr. Wyatt Newman, whose work was the inspiration for my work, and whose expertise, creativity, enthusiasm, understanding, and complete and generous support made this thesis possible. Second, I would like to thank my committee, Dr. Steve Phillips, Dr. Frank Merat, and Dr. Roger Quinn for their technical assistance and time. I would further like to thank Mr. Yuandao Zhang for his invaluable technical assistance.

I would also like to thank Michael Branicky my predecessor, whose work provided me with ideas and inspiration, and whose continued assistance helped the progression of my work. I am also grateful for the inspiration, assistance, and software which was provided to me by my fellow students; Vinay Krishnaswamy, Mark Dorhing, Dave Osborn, and Brian Mathewson.

I am indebted to all my fellow students who made the Mechatronics lab a pleasant place to work. These people include Ashraf Kahn, Greg Glosser, Peter Paul, Ronny Shalev, Tony Buop, Mitch Livstone, Kwok Chung Chan (William), Robert Brunner, John Martens, Dean Velasco, Soheil Sayeh, Lorne

Jenkins, Craig Birkhimer, Nathan Woods, Kamal Souccar, Sean Higgins, Steve Somes, Jonas Olsson, Werner Tseng, Robert Horning, Joseph Cenin, John Murrin, David Sarafian, Alexandru Campean, Jay Patel, Kevin Ballou, and Adam Johnston.

I would like to thank my parents and in-laws for all their invaluable support. Finally, I would like to thank my wife Claudia for her devoted and unconditional support, and my son Jacob for being such a good boy.

This work was made possible through the support of the U.S. Dept. of Energy Sandia National Laboratories under contract #18-4379F, and through the support of CAISR (Center for Intelligent System Research), and CAMP (Cleveland Advanced Manufacturing Program) in Cleveland, Ohio. This work was also supported by NIST (National Institute of Standards and Technology). Their support is gratefully acknowledged.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The word *robot* was first used in 1921 by the Czech playwright, novelist, and essayist Karel Kapek in his satirical drama entitled R.U.R. (*Rossum's Universal Robots*) [24].It is derived from the Czech word *robota*, which literally means "forced laborer" or "slave laborer".

In 1946, George Devol, the acknowledged "father of the robot", developed a magnetic process controller that could be used as a general-purpose playback device for controlling machines. It is generally acknowledged that the "'robot age" began in 1954 when Devol patented the first manipulator with a playback memory. In 1962, General Motors installed the first commercial robot on one of its assembly lines in a die-casting application.

The robots used in industry are usually strictly position-controlled devices. While some processes, especially ones that are repeated and structured, are easily automated using these devices, there are many processes which are not. For robots to be most useful they need to be able to interact with their environment, act autonomously, and maybe be intelligent.

Assuming autonomous, interactive, and inexpensive robots we can easily

imagine numerous fascinating applications of robotics.

1. Robots which are able to successfully interact with their environments using strain gauges, force sensors and compliant motion algorithms, could be used for :

   (a) Materials handling requiring grabbing, pushing, pressing, pulling, holding, rubbing, insertion, scraping, handling of objects of unspecified shapes, positions and sizes, etc..

   (b) Grinding, window washing, painting objects of unspecified shapes or inexact positions, waxing, etc..

   (c) Handling of breakable materials.

   (d) Handling of soft or sensitive materials, like animals and humans.

2. Robots which are able to employ vision systems, or other sensors for object recognition, and from these sensors collect data and generate world models that could be used for navigation, obstacle avoidance, or other types of interaction, could be used for :

   (a) Floor cleaning, lawn mowing, vacuuming, dusting, etc..

   (b) Material and goods transportation.

   (c) Robotic cars with advanced vision-based perception system could replace manual driving, and thus save time, and millions of lifes. The

perception system in this case must be capable of identifying moving and static obstacles (from a moving vehicle), figure out where the road is, identify vehicles and other road obstacles, and to a certain extent figure out distances to obstacles, and velocities of obstacles, and read signs.

3. Voice interactive robots with perception, and compliant control capabilities could be used for a variety of service jobs.

This list could go on for ever. The point is: the autonomous robot[1] is not just another type of automation, or an advanced machine; it is a concept as large as the concept of the machine. Robots will probably continue to be improved and applied to new applications for centuries to come.

However, the quest for the interactive, autonomous and intelligent robot turned out to be a very complex and not very well understood problem. The problem has many aspects, and I will address one of them: if and how reflex-like control methods can be used to achieve robust and autonomous systems.

1.1 Motivation for Reflex Control

Intelligent, autonomous robots require knowledge about themselves and their environment and the ability to integrate this knowledge to make decisions

[1] or the intelligent machine, or the artificial creature

about how to act. Robots need to be able to utilize sensors, interpret the sensor information, and use it as a basis for autonomous action. In nature such abilities were long evolved and built up from robust underlying layers of functionality.

"Evolution has spent most of its time on the essence of being and reacting in a dynamic environment, not on intelligence" (Brooks 1990 [12]). Even very primitive and non-intelligent animals, like insects and mollusks, are able to behave autonomously in complex dynamic environments. These facts show that the efforts to create intelligent robots before true autonomy has been achieved, probably are misdirected. To create creature-like robots, which are able to act independently and perform tasks in the real world, the robot must first be able to react autonomously in a natural environment. Its behaviors should be appropriate to performing the task at hand-as well as maintaining survival.

The simplest form of animal behavior is the reflex. The defining characteristic of a reflex is that the intensity and duration of the response is entirely governed by the intensity and duration of the stimulus [6]. Another similar definition is that the relationship between the stimulus and the response is rigid [1]. Reflexes are used in animals for protective behavior, postural control, withdrawal from painful stimuli, and adaptation of gait to uneven terrain. But reflexes also form a basis for more complex behavior patterns like walking,

running, flying, and jumping.

Reflex-like control strategies have been utilized by a handful of researchers [8, 1, 79, 64, 55, 11] who have found it to be very useful in their robotic application. This fact and the discussion above demonstrates that reflex-like control strategies for robot systems should be further studied and utilized.

1.2 Objective and Thesis Organization

In this thesis I will define, classify and investigate artificial reflexes for robotics. I will give several examples of useful implementations of artificial reflexes to demonstrate the usefulness of artificial reflexes and show how reflex-like control systems can be built. My implementations will demonstrate that artificial reflexes are useful in building autonomous, reliable, and robust robot systems. I will also analyze the performance of reflexive systems and give advice regarding how they should be built and integrated into complex robotic systems.

This thesis is organized as follows.

1. In Chapter 2 I will describe the experimental system I used to implement the different reflex modules and other systems. All experiments were performed on real robotic systems.

2. In Chapter 3 I will give a brief description of the characteristics of reflexes and fixed action patterns in biological systems. I will further give an

overview of earlier implementations of reflex-like control strategies and other behavioral control strategies in robotics. Finally, artificial reflexes are classified and the potential use of the different types of reflexes is discussed.

3. I am using configuration space as a predefined map for my obstacle avoidance reflexes and reflex-based path planning algorithms. In Chapter 4 I will describe my methods for configuration space generation.

4. Chapter 5 contains a description of the "reflexive command filter" and the "C-space inspector" which together comprise an approach to reflexive static collision-avoidance in 4D configuration space. I will explain how the collision-avoidance reflexes are used to protect the robot system from collisions that would result from higher level software errors.

5. Chapter 6 describes so-called reactive trajectory generators which were found to be very useful in the context of reflexive collision avoidance.

6. Chapter 7 starts with an overview of existing robot motion planning methods. The rest of this chapter explains how reflexive collision avoidance can be used in conjunction with higher-level path-planning algorithms and obstacle avoidance methods, and how reflexive collision-avoidance can improve the performance of the system and simplify the construction of such higher-levels. The higher-level path-planning algorithms and

obstacle avoidance methods which are described are :

(a) Guiding Potential Functions.

(b) Quickly computable potential functions.

(c) A fast 4D path-planning algorithm.

7. Chapter 8 contains a description of a few so called "flee reflexes" which are used for on-line dynamic collision avoidance.

8. In Chapter 9 an autonomous sonar-based world mapping system for industrial manipulators is described. This system consists of interacting reflex behaviors and "fixed action patterns"[2].

9. In Chapter 11 I will address performance, stability, cycling, and sampling issues regarding artificial reflexes and the integration of artificial reflexes into complex systems.

10. In Chapter 10 I will give advice on how to analyze and construct systems containing reflexive behaviors using the concepts derived in Chapter 10.

11. Finally in Chapter 12 I will summarize my experiences and discuss the usefulness and feasibility of artificial reflexes in robotics.

[2]See 3 for a definition of fixed action pattern

Chapter 2

Experimental and Development System

The experimental system used in implementing this work consist of a Robotics Research Corporation Manipulator K-2107HR (hereafter RRC robot) and a Sun Microsystems 3/75 Workstation interfaced to a VME-based multiprocessor system. I also used a Hitachi PW-10 in my experiments and I also had the opportunity to use an Adept-I and an Adept-II robot. For my ultrasound-based world mapping system I used Migatronics sonar transducers, and a VME-based sonar processing board developed at Sandia National Laboratories[1], to interface the transducer signals.

2.1 The Computing System

The computing system used to control the RRC robot is shown in Figure 2.1. It consists of a Sun Microsystems 3/75 Workstation and an external VME card cage equipped with single board computers, and 8 Megabyte RAM board, a GPIB interface board or alternatively a BIT-3 board.

[1]The Xycom board

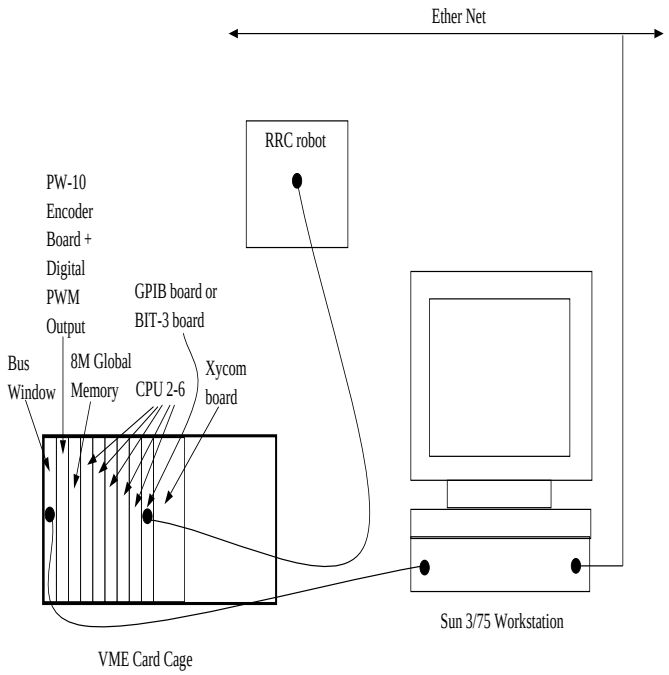


Figure 2.1: The Computing System.

The Sun 3/75 Workstation is interfaced to other workstations and a file server via Ethernet and the external VME card cage through a bus window. The Sun workstation contains a 16.6MHz MC68020 microprocessor, a MC68881 floating point coprocessor, 4 Megabytes of RAM, and a 12 slot VME backplane. The workstation is used for code development for the single board computers, as well as for user interface, diagnostics, and real time graphics while the robot is in operation. The external VME card cage also contains a 12-slot VME backplane. Connected to its common VME backplane are five single board computers. Each of the single board computers contains a 20 MHz MC68020 microprocessor,a MC68881 floating point coprocessor, 1 Megabyte of RAM, and a VME-bus interface. Code developed and compiled on the Sun

is downloaded into the single board computers to run in "real-time" without any operating system overhead. All code was written in the "C" language.

The digital I/O board is used for reading the joint encoders of the PW-10 and transmitting digital commands to the PW-10 amplifiers. The 8 Megabytes RAM board serves as a *global memory* that can be accessed by the Sun as well as the single board computers. The GPIB-interface board reads data from Multibus and transmits it further to the global memory in the VME cage. The BIT-3 board is capable of mapping a small portion of global memory directly to an address space on the Multibus side. The Sandia ultrasound board is used when sonar sensors are utilized. It generates electrical pulses to the sonar transducers and interprets the returning echo.

For the experiments described in this research, the individual computer boards were programmed to execute separate but coordinated tasks. Examples of implementations are :

- Two arm collision avoidance : A combined servo-I/O program for the PW-10 runs on computer board 2. An I/O program for the RRC runs on computer board 3, and a servo program for the RRC runs on computer board 4. The reflexive command filter described in Chapter 5 consist of two submodules, the "active command filter" and the "C-space inspector". The active command filter run on computer board 5 and a C-space inspector on computer board 6. The graphical user interface through

which the operator controls the robot runs on the Sun.

- 4D path planning utilizing reflex control : An I/O program for the RRC runs on computer board 2, and a servo program for the RRC is running on computer board 3. Active reflexes run on computer board 5 and a C-space inspector on computer board 6. A path planning algorithm runs on computer board 4. The graphical user interface through which the operator controls the robot runs on the Sun.

- Sonar-based world mapping system. An I/O program for the RRC runs on computer board 2, and a servo program for the RRC is running on computer board 3. Active reflexes are running on computer board 5 and a C-space inspector on computer board 6. A sonar data analysis program runs on computer board 4. The graphical user interface through which the operator controls the robot runs on the Sun. The rest of the system modules run on the Sun 3/75.

2.2 The RRC Robot

The RRC robot is a seven degree-of-freedom kinematically redundant manipulator. The repeatability of the RRC robot is extremely high. It has DC motors and harmonic drives. The RRC robot has resolvers and strain gauges mounted at the transmission output. It weight 500 lbs and has a payload capacity of

4 lbs. It is intended for research and space exploration. A photograph of the RRC robot in its natural mechatronics environment is shown in Figure 2.2.

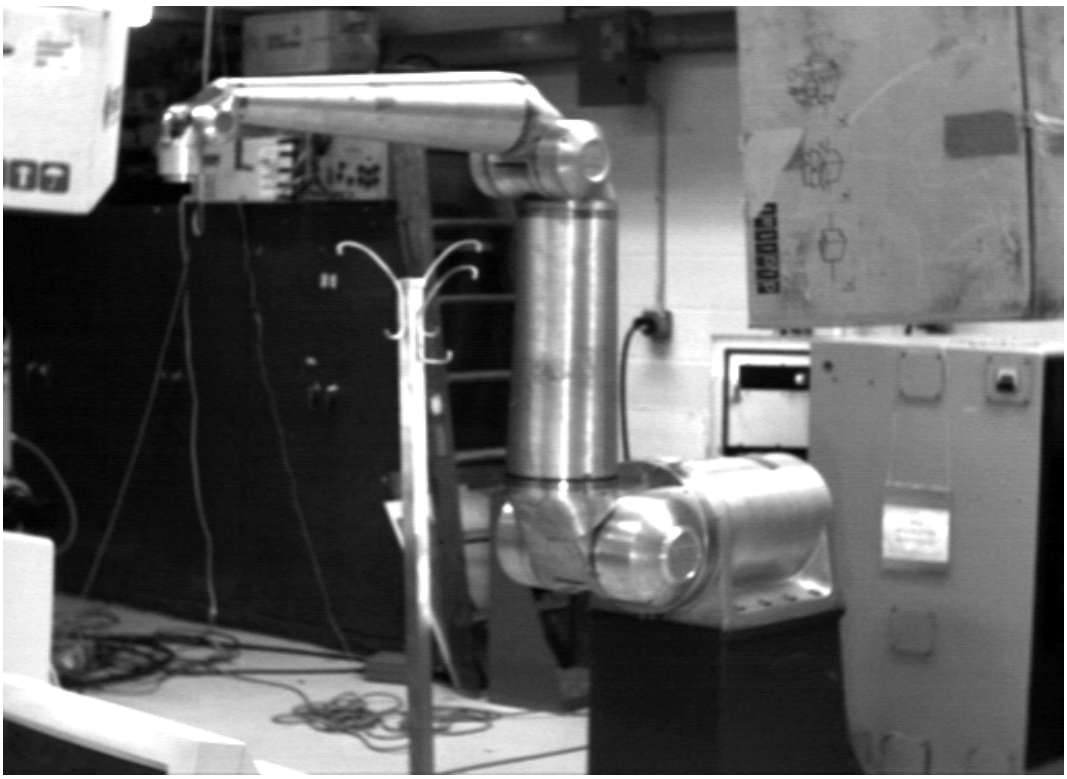


Figure 2.2: The RRC robot.

The RRC robot was originally equipped with an independent PID-controller for each joint, and was controlled by algorithms running on Intel processors in an open multibus-based architecture. Work was done by Mark E. Dohring to connect the RRC robot to the VME-bus environment in the mechatronics lab. A program written in the PLM/86 programming language runs on the Intel processor and reads the A/D converters, the resolvers and other sensor data and transmits the data to the VME-cage where one of the processor boards converts the digitized data is into SI-units.

Joint	θ_i	α_i	d_i	a_i
1	θ_1	$90°$	0	0
2	θ_2	$90°$	0	$a2$
3	θ_3	$-90°$	$d3$	$a3$
4	θ_4	$90°$	0	$a4$

Table 2.1: DH-parameters for the RRC robot.

The communication between Multibus and the VME-bus was originally a two step process in which the Multibus sent data to a GPIB interface which in turn sent the data to the VME-bus. Now the Multibus sends data directly to the VME-bus. This is made possible through a BIT-3 interface. The BIT-3 interface is capable of mapping a small portion of the Global Memory board in the VME-cage directly to Multibus.

The seven possible joint rotations of the RRC robot are displayed in Figure 2.3. The kinematic redundancy makes the RRC robot very interesting with respect to obstacle avoidance. The extra degree of freedom allows the robot not only to position its tool arbitrarily in space, but also allows it to position the robot body so that obstacles can be avoided.

The joint coordinate systems I chose are shown in Figure 2.4, and the corresponding Denavit Hartneberg parameters are shown Table 2.1. The values of θ_i in the configuration shown are given in Table 2.2. A detailed overview of the robot kinematics and the corresponding matrixes are given in appendix A.

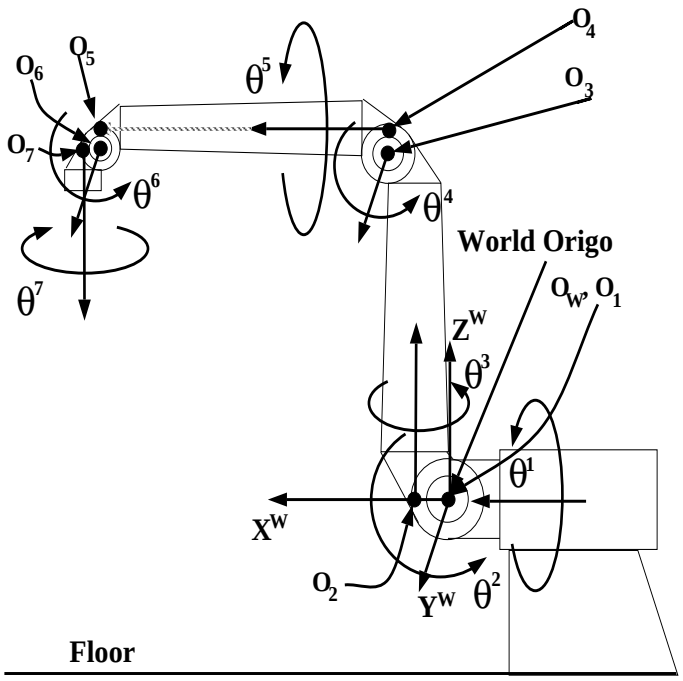


Figure 2.3: The joint rotation of the RRC robot, and the world coordinate system.

$\theta_1 = 0°$
$\theta_2 = 90°$
$\theta_3 = 0°$
$\theta_4 = 90°$

Table 2.2: The home angles of the RRC robot

2.3 Conclusions and Overview for Chapter 2

The experimental system used in implementing this work consist of a Robotics Research Corporation Manipulator K-2107HR and a Sun Microsystems 3/75 Workstation interfaced to a VME-based multiprocessor system. The sonar-based world mapping system used Migatronics sonar transducers, and a VME-based sonar processing board developed at Sandia National Laboratories[2] to

[2]The Xycom board

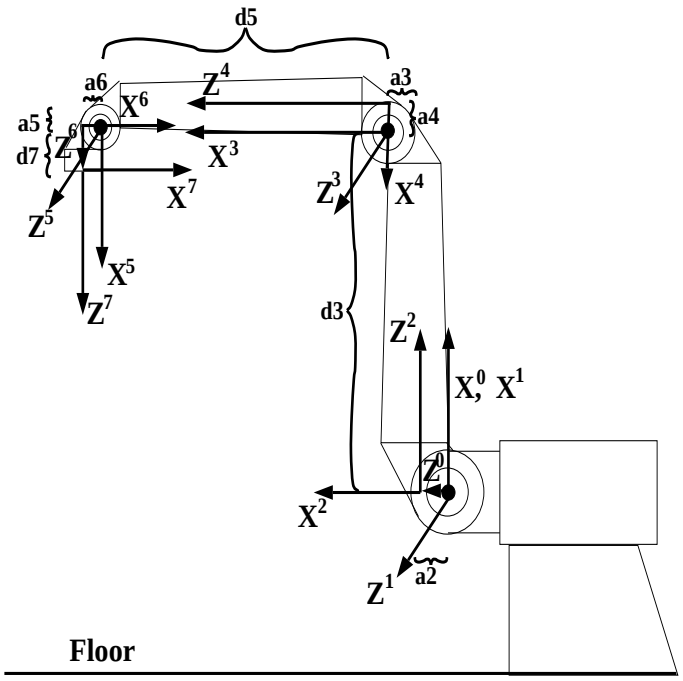


Figure 2.4: The joint coordinate systems of the RRC robot and the corresponding parameters.

interface the transducer signals.

Code developed and compiled on the Sun is downloaded into the single board computers to run in "real-time" without any operating system overhead. However, it is only possible to run one application at a time on a specific processor-board. A typical application running on this system is the two arm collision avoidance scheme:

- A combined servo-I/O program for the PW-10 runs on computer board 2. An I/O program for the RRC runs on computer board 3, and a servo program for the RRC runs on computer board 4. The reflexive command

filter described in Chapter 5 consist of two submodules, the "active command filter" and the "C-space inspector". The active command filter run on computer board 5 and a C-space inspector on computer board 6. The graphical user interface through which the operator controls the robot runs on the Sun.

Unlike a system based on Lynx or Vxworks this system has very few features. You cannot read directly from the file server disk to the processor boards, you don't have multi-tasking, you cannot set a fixed loop frequency for your applications, etc. However, you do not have any operating system overhead and the system is inexpensive. Each application program runs asynchronously on a dedicated processor board at maximum speed.

Chapter 3

Reflexes in Biology and Robotics

In this chapter I will give an overview of reflexes in biology, behavioral robot architectures, and the history of reflex control in robotics. I will briefly describe a functional simulation of a locust locomotion system. Finally I will classify artificial reflexes for robotics.

3.1 Reflexes and Fixed Action Patterns in Biology

Reflexes have been studied in many context, and in many animals. These studies demonstrate that reflexes provide autonomous, multiresponsive systems, which functionality to a large extent can be understood and simulated. Detailed descriptions of the neuron connections and their functionality have for example been done in the case of the walking locust [16, 21, 18, 17, 19]. In the case of the walking locust there are basically two types of reflexes involved in the locomotion of the leg : avoidance reflexes and resistance reflexes. The avoidance reflex moves the leg away from mechanical stimulation of external

sensors of the leg, with the precise form of movement depending upon the spatial location of the stimulated receptors. The resistance reflex is initiated by internal sensors, like pressure, tension and knee angle sensors, and ensures that an imposed movement of a joint is resisted by an opposing muscular force. The two types of reflexes oppose each other to give rise to a leg motion, and a final relaxation of the leg [16]. In [6, 26, 7] a biologically inspired architecture for locomotion in a hexapod robot.

However, the intensity and the sign of reflexes can also change depending upon internal factors [6]. For example the local reflexes of one leg in a locust can only occur if the positions and movements of other legs are appropriate at that time. Intersegmental intern neurons, which are neurons mediating signals between segments of neurons, mediate the afferents from other legs, and gates local reflexes through the neurons that mediate the local reflex [20]. An important thing to note is that animals typically respond to a small subset of the total amount of sensory information [1].

Fixed-patterns are a somewhat more complex form of behavior. In [6] a *fixed action pattern* is defined as an extended, largely stereotyped response to a sensory stimulus. Fixed action patterns can be seen as sets of reflexive behaviors triggered by each other. In Section 9 I will give an example of how reflexes and fixed action patterns together can generate more complex robot behavior.

3.2 Behavioral Robot Architectures and Implementations of Artificial Reflexes in Robotics

Because of the problems traditional AI has had in implementing fast functioning, autonomous systems in noisy, unpredictable environments, and with sensors giving incorrect information, scientists have been looking into alternative architectures for robots. These new architectures have much in common with biological systems. Examples of such architectures are Rodney A. Brooks' famous subsumption architecture [12, 13, 15, 29, 30], Rajko Tomovics reflex-controlled prosthesis [8, 71, 72, 70], and situated-automata-models [35], situated agents with goals [51], and animal behavior-based architectures [6, 1] etc.

Brooks argues in [12] that "Artificial intelligence research has foundered in a sea of incrementalism". No one is sure where to go improving on earlier demonstrations of techniques in symbolic manipulation of ungrounded representations. The traditional approach has emphasized the abstract manipulation of symbols, whose grounding in physical reality has rarely been achieved. R.A. Brooks argues further that the "symbol system hypothesis" upon which, "classical" AI is based, is fundamentally flawed. In classical AI none of the modules are generating the behavior of the total system, only specific combinations of them. In what Brooks calls novelle AI, each module by itself

generates a behavior. Novelle AI, is according to Brooks, founded upon the "Physical grounding hypothesis". Brooks defines the *Symbol system hypothesis* as "Intelligence operates on a system of symbols" and the *Physical grounding hypothesis* as "To build a system that is intelligent it is necessary to have its representation grounded in the physical world".

Traditionally mobile robot control has been decomposed for synthesis based on information processing functions. Brooks' approach is to have task-achieving behaviors as his primary direction for decomposing the control system [15]. Brooks' subsumption architecture basically consists of layers of control systems incrementally built to let the robot operate at increasing levels of competence [13]. Each layer provides an independent behavior, and its output is a direct result of some properties of the sensors connected to that layer. Higher layers of control can inhibit lower layers of control. There is no need for a central control module, and it is possible to incrementally insert more layers to increase the robot's competence. Each layer is built from a set of small processors which send messages to each other. The processors run completely asynchronously, monitoring their input wires, and sending messages on their output wires [13]. The lowest layer in the subsumption architecture employs reflex control for obstacle avoidance [15]. Brooks has very successfully implemented his architecture on several mobile robots: Herbert, which navigated in office buildings and stole Coke cans [12, 29]; and Squirt a miniature robot

which demonstrated autonomous behavior [29, 30].

Leslie P Kaelbling and Stanley. J. Rosenschein have been working on the so called situated-Automata-model, which models the world as a pair of interacting automata, one corresponding to the physical environment and the other to the embedded agent. An embedded agent is defined as a directly reacting agent which is continuously informed about the environment. The embedded agent consists of a perception component which delivers information, and an action component which maps this information to action (information-action-pair). The action components are not only functions of information but also functions of the goals the agent is pursuing for the moment. These goals are extracted from higher order goals with the help of information. For example "robot hungry" can be extracted to "find apple in right drawer", from the fact that there is an apple in the right drawer. They constructed a language, *Gapps*, that generates run time programs which are reactive, does parallel actions, and carries out strategies made up of very low-level actions, based on this approach [35].

Pattie Maes suggests an architecture which successfully combines the direct coupling between perception and action with goal orientedness. Her system consists of competence modules, which, if certain preconditions are fulfilled, become active to a certain degree. What activation level a competence module

is set to depends on the currently observed situation consisting of all propositions that are currently observed by their associated virtual sensor, the global goals the module can achieve, and the activation levels of so called "successors" and "predecessors". If a competence module's action pattern is such that it naturally should be followed by another module's action pattern, the latter is a successor of the first. If a competence module's action pattern is such that it is desirable, but for the moment not executable, it raises the activation level of its predecessors. If the activation level of a competence module fulfills certain conditions (thresholds etc.) it takes control and performs its task [51].

Randy D. Beer, Hillel Chiel and Leon S. Sterling constructed a biologically insprired architecture locomotion in a hexapod robot. They used a simplified neuronal model for the locomotion, and replaced the neurons in that model with an often used electronic model for neuron compartments [6].

Rajko Tomovic is the first person to use the concept of reflex control. Already in 1962 he constructed a robotic hand which behaved in a reflex-like manner. His hand showed unusual mechanical simplicity and flexibility and the hand was controlled electrically instead of through complex mechanical designs. More importantly there was direct feedback between the pressure transducers and the motor force. The fingers would close around arbitrarily shaped objects in a reflexive manner [71]. Tomovic improved his hand and

his sensors and control methods [72], and later developed a reflex-based architecture which he applied to artificial limbs like legs and hands [70, 8]. In his reflex-based architecture, pattern-recognizing modules monitored bionic sensors and set Boolean variables which were used to set the limb in different states and in which different transition trajectories were generated.

Tracey L. Anderson and Max Donath used animal behavior as a basis for robot design. They found in their studies of animals, that many types of behaviors and especially reflexes often use a very small part of the provided sensory information. They further found that the behaviors are often independent of each other and sometimes conflicted with each other. They implemented reflexive behavior on a mobile robot called scarecrow. Examples of artificial reflexes they implemented were :

- Avoidance behaviors :
  - A halt function for collision with static objects.
  - A repulsive force function for moving obstacles.
- Attraction behaviors
  - A location attraction reflex which looks for desired locations.
  - A forward attraction reflex which generates commands resulting in motion along the current orientation of the robot.

- Object following and attraction reflexes.
- Open space attraction reflexes.

The reflexes generated potentials which were combined into a resulting force which was further fed as a command to the motor [1].

Another very interesting implementation is the reflex control for slippage and contact in the prototype leg for the Automatic Suspension Vehicle developed by Ho Cheng Wong and David E. Orin [79]. The reflex control for the prototype leg is supposed to be used when there is an abrupt change in leg-environment interaction. The reflex control is based on minimal knowledge of the environment, it overrides higher-level commands, and it executes quickly. They modeled the entire walk cycle as; lift-off; transfer; placement; contact, and support. Examples of situations in the walk cycle, where reflex control is useful are :

- There is no firm foothold before the kinematic limits are exceeded, which implies that the leg stepped into a hole or a ditch. This might occur in the placement phase.
- Unexpected motion when the foot comes in contact with an obstacle. This might occur during the transfer phase.
- When foot contact takes place the transition trajectory must change abruptly and the foot must be protected from damage. This takes place

during the contact phase.

- Abrupt changes in velocity and force in the support phase indicate foot slippage.

In their analysis of their reflexive control architecture they asserted that reflexes consist of three phases: initiation of the reflex; the transitory control of the reflex; and the completion of the reflex. If during the support phase, a non-zero foot velocity and an abrupt actuator pressure change is detected, this would indicate slippage, and the slippage reflex would be initiated. During the transitory control phase in the slippage reflex, the commanded tangential force is quickly reduced. The completion of the reflex takes place when the velocity is below a certain threshold value and the lift actuator error is small.

Another Reflex application was implemented by David W. Payton on the Autonomous land vehicle [64]. He describes a control hierarchy based on immediacy. The higher-level modules perform tasks involving time consuming assimilation, while the lower level modules perform tasks requiring the greatest immediacy. The layers of the control system in his model are : mission planning, map-based planning, local planning with high level maneuvering, and reflexive planning. Connected to the control layers is also a layered perception system. The reflexive planning module consist of a large selection of expert sub-modules, each of which is capable of making decisions about vehicle actions under specialized circumstances. All reflex modules are divided into

two distinct elements: a perceptual component called a virtual sensor and an action component called a reflexive behavior. He defines a virtual sensor as a simple sensing and processing unit which provides assimilated sensor data. Reflexive behaviors are highly procedural units due to the high demands for immediacy. Examples of reflexes he implemented were :

- Slow for obstacle.
- Turn for obstacle.
- Maintain heading.
- Follow left edge.

The concept of Reflex control for obstacle avoidance for industrial robots was introduced by Wyatt S. Newman in his Ph.D thesis [56]. Reflex control for obstacle avoidance has since been further developed and implemented on different robots by Dr. Wyatt Newman, Michael Stephen Branicky and the author [55, 54, 9, 58, 11, 76, 77, 75, 78, 74, 57, 59]. This *reflex controller* exhibits the following virtues: It does not suffer from unrealistic or overly restrictive assumptions on robot dynamics; it fits into control hierarchies and does not interfere with normal actions unless imminent danger is present; its complexity does not increase with higher dimensions; it does not fail or slow down as environment complexity increases; it does guarantee collision avoidance (at least for static obstacles) and can be computed on line; and in

certain implementations it is fail-safe i.e. if the reflex controller quits or is too slow this will still not result in a collision. When the reflex controller easily can be incorporated with other control schemes with minimum non-essential influence on higher-level controls, it serves as a natural building block in more advanced obstacle avoidance and path planning schemes.

3.3 A Classification of Artificial Reflexes

Reflexes are typically simple, short in duration, and directly tied to some proprioceptive or exteroceptive sensory input. In [64] artificial reflexes are modeled as consisting of two components, a perceptual component called a virtual sensor and an action component called a reflexive behavior. I will also model artificial reflexes this way. The virtual sensor monitors the environment and the robot state and detects certain easily defined conditions and generates an input to the active component.

I will further classify artificial reflexes depending on whether they are implemented in parallel or in series. Artficial reflexes which are implemented in series with the control structure are referred to as reflexive filters. Artificial reflexes which are implemented in parallel with the control structure will be referred to as reflexive actions. A reflexive filter is implemented in series with the control system, which means that all higher-level commands must pass

through the reflexive filter. A reflexive filter would typically be used to protect the system from higher level commands which would result in undesirable results. A reflexive filter could also be used to provide a guaranteed low-level behavior for other reasons. A reflexive action on the other hand has no effect on the system until shortly after it has been activated. Another important distinction is whether the reflex module depends on higher-level commands or not. The different cases are listed below and illustrated in Figure 3.1.

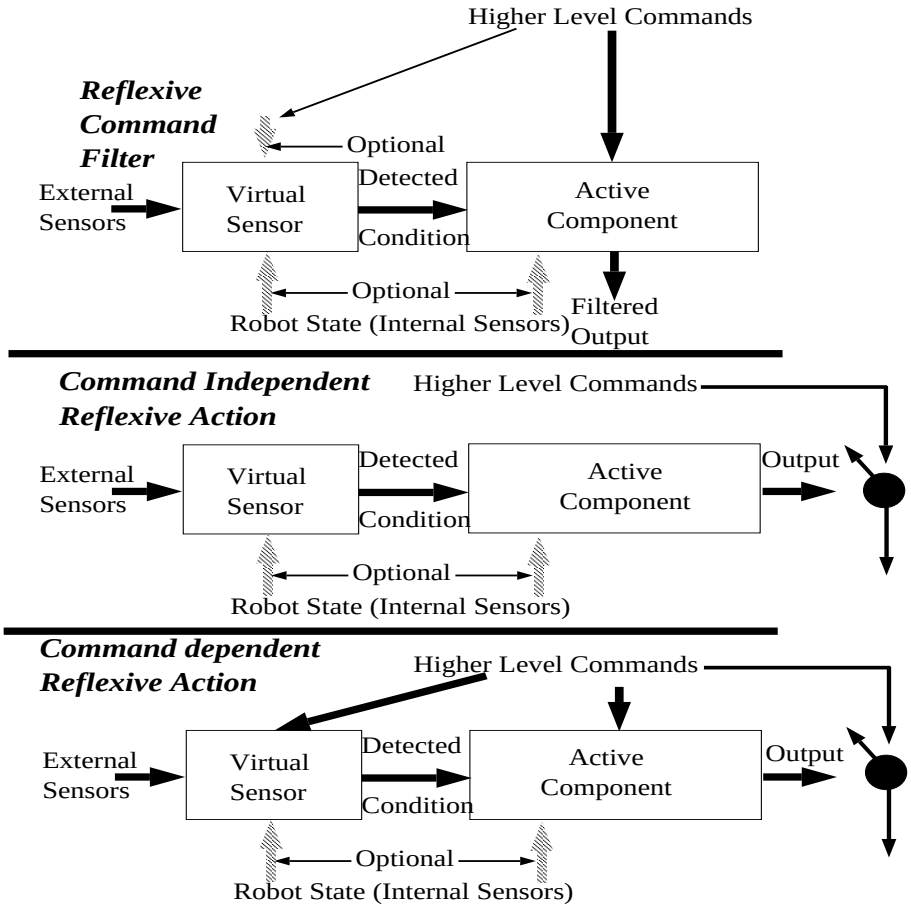


Figure 3.1: A modular description of a reflex action, a command dependent reflex action, and a reflexive command filter

1. The artificial reflex is implemented in parallel with the control architecture, in other words the reflex controller generates commands into the system when its virtual sensor detects certain conditions. Further the active component does not depend on the current commands generated by higher-level controls in the control structure. This type of reflex control is typical in biological systems, and common also in robotics. This type of reflex can be used for protection, withdrawal and attraction, postural control, and as a component of fixed action patterns or complex behaviors. I will refer to this type of artificial reflex as a *command independent reflexive action.*

2. The artificial reflex is implemented in parallel with the control architecture, and the active component depends on the current commands from higher-level controls in the control structure. This type of reflex control generates commands into the control structure depending on the currents commands. The virtual sensor would typically restrict higher-level commands when certain conditions are detected. A watchdog which detects failures in higher-level modules and replaces commands coming from these modules would be such a reflex. Another example is a reflex which replaces all commands that would result in the robot getting closer to a danger zone. I will refer to this type artificial reflex as a *command dependent reflexive action.*

3. The artificial reflex is implemented in series with the control architecture, in other words commands generated by higher-levels of control are continuously monitored and approved or replaced by the reflex controller. This type of artificial reflex acts like a command filter and is for that reason called a *reflexive command filter*. The virtual sensor in the reflexive command filter detects certain conditions and derives from those restrictive conditions which are put on the incoming commands. The reflexive command filter is used when it is essential that all higher-level commands are monitored and approved. In other words, all commands must fulfill the restrictions put on them by the environmental circumstances detected by the virtual sensor. The reflexive command filter can be used for fail-safe collision prevention, regulating force contact, protection against higher-level software errors. In the case of fail-safe collision prevention the virtual sensor would provide the active component with a subset of obstacle free space which includes the current configuration. The active component would approve or replace higher-level commands in such a way that it always guarantees that the robot stays within this space. In Section 5 I will describe our reflexive command filter for obstacle avoidance. The virtual sensor in this case is called the "C-space inspector". In the case of contact forces the virtual sensor would generate restrictions related to the apparent passivity of the robot.

4. For a reflex to be implemented in series with the control structure it must at least depend on higher-level commands. For this reason artificial reflexes which do not depend on incoming commands cannot exists.

The distinction between reflex modules, and other behavioral modules, as implemented in parallel and series, results in a general system architecture given in Figure 3.2. In this hierarchical structure the behavioral command filters form a "stem" through which all commands are passed and filtered. The behavioral action modules are organized in layers. Within each layer the different modules compete for control. These modules may depend on external sensors, and higher level commands, as well as the robot state. These potential dependencies are not indicated in Figure 3.2. The "virtual switch controller" in Figure 3.2 is not necessarily a separate switch controller. In the systems I implemented, the virtual switch controller is simply the result of the interaction among the virtual sensors of the behavioral modules. In other words the individual modules take control of the "virtual switch controller", depending on the state of the virtual sensors. However, it is conceivable that the virtual switch controller represents a separate module, or a human operator. This structure will in the be referred to as the pine-tree structure in the continuation.

I will also define the concepts of simple reflexes, triggered reflexes, containment reflexes, and repulsor reflexes.

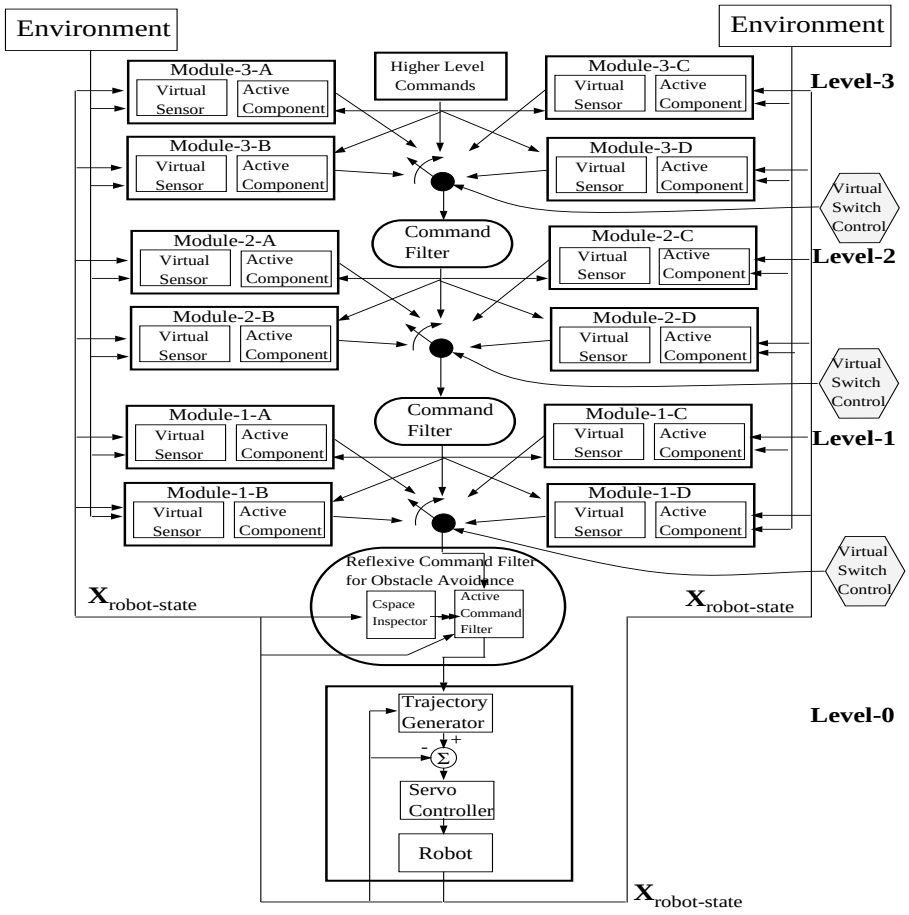


Figure 3.2: An illustration of a generic behavioral structure. The behavioral command filters form a "stem" through which all commands are passed and filtered. The behavioral action modules are organized in layers. Within each layer the different modules compete for control. The virtual switch controller represents the interaction among the modules.

- Simple reflexes are defined as reflexes which are active in a specific region of the state space called the activation region of the reflex. It should be noted that the activation region might change due to external stimuli, or a changing environment. However, the activation region is not dependent on the current robot state, or internal task space commands.

- Triggered reflexes are reflexes which are activated if the robot enters a certain region of the state space called the trigger region, and remains active as long as the robot remains in a certain region of the state space called the activation region. The activation region must be a superset of the trigger region. In other words the activation region has hysteresis. The difference between the activation region and trigger region is called the hysteresis region. It should be noted that if the trigger region and the activation region are identical the reflex is a simple reflex. This type of reflex is illustrated in Figure 3.3.

- A containment reflex is a reflex for which all commands generated by the reflex are confined to the same region in which the reflex is active. A containment reflex can for example be a simple, or a triggered reflex. Examples of a simple-containment reflexes are:

  1. Reflexes which generate commands which bring the robot to a specific point.

  2. Reflexes which make the robot converge to a subset, rather than a specific point.

  3. A reflex which keeps the robot in the position within the activation region where the robot entered the activation region (was first found within the activation region). This particular type of reflex is called a

"hold reflex". The point where the reflex "holds" the robot does not have to be located on the boundary of the activation region. External stimuli can for example create a new activation region. Sampling times and delays can also allow the robot to penetrate a fixed activation region.

4. Reflexes which confine the robot to the activation region, like the static command filter (general case).

Figure 3.4 illustrates these four simple-containment reflexes.

- Repulsor reflexes push the robot outside the activation region. A repulsor reflex can be a simple, or a triggered reflex.

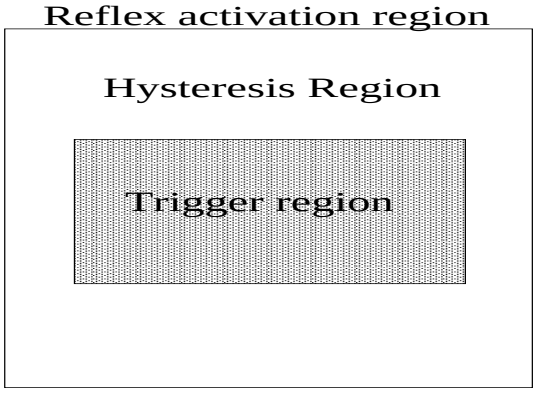


Figure 3.3: If the robot enters the grey area the reflex is activated. This is called the trigger region. The reflex is not deactivated until the robot leaves the white area. The white area is called the hysteresis region. The activation region is the union of the hysteresis region and the trigger region.

These concepts will be further discussed and used in Chapter 10.

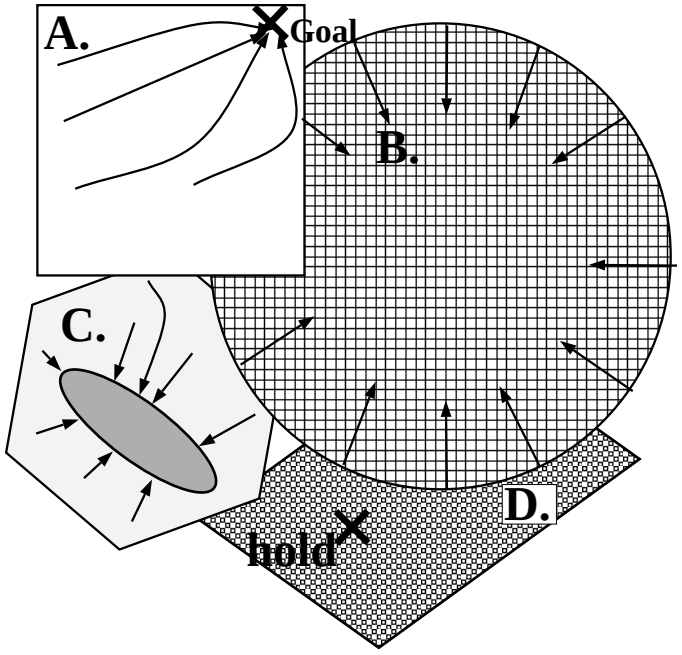


Figure 3.4: Four reflexes R_A, R_B, R_C, and R_D are active in four different regions. R_A generates a specific command, R_B keeps the robot within the activation region, R_C makes the robot converge to a subset of the activation region, and R_D holds the robot where it entered the activation region.

3.3.1 A Classification of the Reflexes Described in this Thesis

In this section I will attempt to classify the reflexes and fixed action patterns I implemented, according to the classification scheme described above. I will also identify the virtual sensor and active component for each reflex, and give a compact description of the basic operation and general purpose of each reflex.

- *The reflexive command filter for obstacle avoidance.* This reflex is described in Section 5. The reflexive command filter is command dependent, and implemented in series with the rest of the system, and is for

this reason of the reflexive command filter type. The virtual sensor in this case is called the C-space inspector, and the active component is called the active command filter. The C-space inspector provides the active command filter with an obstacle free prism, while the active command filter uses the free-space prism to determine whether a command is safe, or not.

The purpose of this reflex is to protect the robot from collisions that would occur if higher level commands were executed. It prevents collisions that would occur if erroneous higher-level software or operator commands were carried out, without otherwise interfering or effecting the system.

- The startle reflex is a component of the flee reflex described in Section 8. Two types of startle reflexes are described in this thesis. In the case of the wall-emulating startle reflex, the moving obstacle is implemented as a moving wall, while for the block-emulating startle reflex, moving obstacles are modeled as occupying static blocks of work-space. This is a very quick emergency reflex which generates an easily computed response in the direction away from an approaching or potentially approaching obstacle. The startle reflex is fired only once and the generated flee direction is independent of the current command. This is an example of a reflexive action.

  The virtual sensor of the startle reflex monitors two conditions:

1. The existence of a virtual wall at the current position, or an approaching virtual wall nearby the robot. In the case of the block-emulating flee reflex the virtual sensor for the startle reflex detects blocks in the world space which are occupied by an obstacle and sufficiently close to the robot.

2. It checks whether another flee reflex, like the retraction reflex or the hold reflex, are active. If that is the case the startle reflex is inhibited.

The active component of the startle reflex computes a flee direction using the Jacobian, and delivers the resulting commands to the servo-controller.

- The retraction reflex is another component of the flee reflex. This reflex generates commands which make the robot flee in a desired flee direction, away from approaching or potentially approaching obstacles. The flee direction is independent of the current command, so this is yet another example of a reflexive action. Again two types of retraction reflexes are described in this thesis. In the case of the wall-emulating retraction reflex, the moving obstacle is implemented as a moving wall, while for the block-emulating retraction reflex moving obstacles are modeled as occupying static blocks of work-space.

  The virtual sensor detects the existence of a startle reflex, and the active component generates a flee command computed from a flee potential and

the inverse Jacobian using configuration coordinates.

- The hold reflex is a component of the block-emulating flee reflex. This reflex keeps the robot at a safe distance from a moving or unanticipated obstacle. The safety distance decreases by time if the obstacle remains static. If a higher-level control module generates a command which would bring the robot further away from the obstacle, this command is approved by the hold reflex. This is an example of a command-dependent reflexive action.

  The virtual sensor of the hold reflex detects the existence of the retraction reflex and monitors the distance to the obstacle in question. The hold reflex becomes active when the retraction reflex has been initiated and the distance from the obstacle is large enough. It is inhibited if the distance from the obstacle is even larger. The active component either approves higher-level commands or replaces them with commands that keep the robot in its current position.

- The halt reflex is one of the modules in the sonar-based world mapping system described in Section 9. This reflex generates commands to the servo-controller which halt the robot and keep it in its current position for a certain amount of time. The purpose of this reflex is to halt or slow the robot down when new obstacles are detected or when the robot is

performing large moves. The halt reflex is of the reflexive action type.

The virtual sensor detects new obstacles and large uninterrupted robot motions. The active component holds the robot in the position where the the robot was located when the halt reflex was initiated. The halt reflex is inhibited to a certain degree when the work space is considered to be well known by the sonar-based world mapping system.

- The fixed action patterns used in the sonar-based world mapping system are all reflex-like in the sense that they are simple, low level and they can be classified and modeled the same way reflexes can. However, in this case there is not a direct relationship between stimuli and response. The time duration of fixed action patterns is in general much larger, and they play an active part in the generation of the overall behavior of the robot. They further generate a set of commands rather than a single response.

  The fixed action patterns described in Section 9 are all similar to reflexive actions in their general structure. The virtual sensor active component pair can in each case be described as :

  1. The virtual sensor for Look-Around checks if the robot has a task to perform, or whether it is idle. The active component generates robot motion for the purpose of finding unmapped obstacles.

2. The virtual sensor for Look-Path checks if the robot has a task to perform and is currently moving. The active component generates the appropriate robot tool motions for looking for obstacles in the robots path.

3. The virtual sensor for Beam-At detects inconsistencies with the current world map. The active component, in this case, generates commands that will direct the sonar sensors in the direction of the found inconsistency, for all "robot body" configurations.

4. The virtual sensor for Approach detects inconsistencies with the current world map. The active component moves the robot slowly towards the location of the detected inconsistency.

5. The virtual sensor for Investigate monitors the distance to the found inconsistency, and how well the inconsistency has been investigated. The active component generates a motion scheme which will place the robot in multiple positions around the inconsistency. Investigate is inhibited when the inconsistency has been well explored.

6. Make-Map is a higher-level behavior which still can be modeled as a virtual sensor and active component pair. The virtual sensor for Make-Map monitors the confidence level thresholds of the investigated space to determine if a map should be done or not. The active component makes a map using retrieved sonar data.

This list demonstrates how reflexes can be classified, but it also indicates how artificial reflexes should be constructed. When constructing a reflex you should :

1. Identify the purpose of the reflex, and determine if a reflex module is the appropriate choice for solving a particular problem. If the desired robot behavior should be normally non-active or transparent, and active only under certain conditions, and further if, the desired behavior is simple and low level, a reflex module is appropriate.

2. Determine whether there are any higher-level commands which need to be monitored by the reflex module. Determine whether the reflex module should be implemented in series or in parallel. If there are higher-level commands that must be continuously monitored and "approved", the reflex should be implemented in series, and otherwise in parallel.

3. Determine under which conditions the reflexive behavior should be active. These conditions should be detected using a virtual sensor module.

The list above also demonstrates that some complex behaviors, particularly fixed action patterns, can be modeled the same way reflexes are modeled.

3.4 Conclusions and Overview for Chapter 3

This chapter gave an overview of reflexes in biology and robotics. It was mentioned that reflexes are defined as simple responses which are short in duration and directly tied to sensory input, and that fixed action patterns are defined as extended stereotyped responses to sensory stimulus. It was also mentioned that biological reflexes provide autonomous, multiresponsive systems, which functionality to a large extent can be understood and simulated. It was mentioned that detailed descriptions of neuron connections and their functionality has been given.

The traditional approach to artificial intelligence in robotics has emphasized the abstract manipulation of symbols, whose grounding in physical reality has not been achieved. This approach has failed to create "intelligent robots" or "autonomous robots". For this reason, behavioral approaches to intelligent control have become more popular. A few of these behavioral approaches were discussed in Section 3.2, a few of which utilized reflex-like control.

It was concluded that it was convenient to model artficial reflex modules as consisting of two submodules, a virtual sensor and an active component. Artificial reflexes are either implemented in series, in which case they are called reflexive filters, or in parallel, in which case they are called reflex actions. The same is true for other types of behavioral modules. This leads to the architecture shown in figure 3.2. I also classified reflexes according to whether

they were dependent on higher-level commands or not. This chapter also introduced the concept of simple containment reflexes, repulsor reflexes, and simple triggered reflexes.

The motivation behind this chapter was to give an introduction to how reflex-like control has been used in robotics reasearch, and to show that reflex control is a research area with significant potential.

Chapter 4

Configuration Space Generation for the Purpose of Reflex Control

The reflexive command filter to be described in Section 5 relies on a predefined obstacle map which in our case is a configuration space map. The concept of configuration space was first used in [47]. The configuration space of a robot $\mathcal{A}$ is the space $\mathcal{C}$ of all configurations of $\mathcal{A}$. For a manipulator arm, the configuration space could be the joint space. Every obstacle β_i in the work space[1], maps in $\mathcal{C}$ into a region $\mathcal{C}(\beta_i) = \{\forall \mathbf{q} \in |\mathcal{A}(\mathbf{q}) \cap \beta_i \neq \mathbf{0}\}$ Where $\mathcal{A}(\mathbf{q})$ denotes the subset of the workspace occupied by $\mathcal{A}$ at configuration $\mathbf{q}$. The union of all $\mathcal{C}(Obstacles)$ is called the $\mathcal{C}(Obstacle)$ region, and its complement is called the free-space. In configuration space each pose of the robot is a single point. The dimension of the configuration space is the number of the parameters representing a configuration. In the remainder of this thesis the configuration space will be referred to as the *C-space*.

According to [34] there are seven standard methods available to compute

[1] With work space is meant the subset of the physical 3D space which is reachable by the robot.

configuration obstacles:

- Point evaluation: The robot is placed in all possible configurations and it is determined whether the robot is intersecting work space obstacles or not. This method is simple and applicable to all types of robots. It is, however, very time consuming.

- In the boundary equation method, one derives the constraints of the configuration variables that bring the robot in contact with obstacles. The contact constraints define the boundaries of the configuration obstacles. This method is very hard to use in high dimensions and for complex obstacle and robot shapes.

- Minkowski set difference: The Minkowski set difference of two sets $\mathcal{A}$ and $\mathcal{B}$ are the set of points $\mathbf{Mdiff}(\mathcal{A}, \mathcal{B}) = \{a - b | a \in \mathcal{A}, b \in \mathcal{B}\}$. For a rigid object without rotation, the configuration obstacles are the union of Minkowski set differences between areas occupied by obstacles and the robot. In Figure 4.1, the reference point of the robot cannot be placed in the shaded region, which is $\mathbf{Mdiff}(obstacle, robot)$. This method is primarily used for polytopes.

- The needle method: All but one of the configuration parameters are fixed, and the values of the variable parameter that bring the robot in contact with all the obstacles are computed.

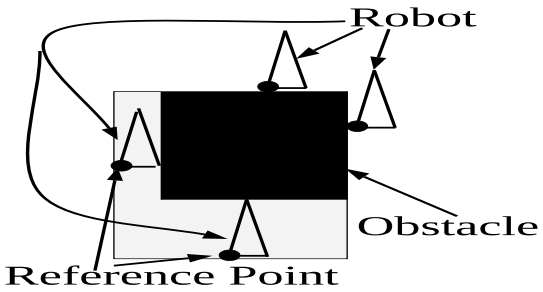


Figure 4.1: The shaded region is the Minkowski set difference. The reference point of the robot in this orientation cannot be placed in the this region.

- The sweep volume method computes the volume in the world space generated by a robot as the robot configuration is varied over a set in the C-space. If the volume does not intersect any obstacle, the C-space set is declared free space and otherwise further investigated. This method is very efficient, but the sweep volume is hard to compute if the set has a high dimension or the robot links have complex shapes.

- The template method computes the configuration obstacles due to features of world obstacles. These features could for example be lines or points, and the corresponding configuration obstacles are called templates. The world obstacles are represented as a union of features for which templates are computed. The configuration obstacles are then computed by "stamping" the corresponding templates into C-space. This method was developed in [59].

- The Jacobian-based method : The Jacobian of a robot is used to relate the displacement, δx, of a point on the robot to the corresponding change

in the robot configuration. These displacements are used to relate a C-space "chunk" to a work space "halo" around the robot. This "halo" is used to check for obstacle intersections. This method was developed in [62].

In the case of the reflexive command filter described in Section 5 the configuration space contains static obstacles, corresponding to joint limits, static obstacles in the robot's environment, and poses representing self-collision configurations. These C-space obstacles usually do not require recomputation or sensory input and can therefore be precomputed. However, sometimes obstacles are brought in or out of the robot's workspace, or mapped in using sensors (for example ultra-sonic sensors). In these instances the C-space map must be recomputed as fast as possible. For this purpose, I constructed a configuration-space generator that takes workspace obstacle positions and their sizes as inputs, and rapidly adds them to the C-space map. The configuration-space generator uses the template method described above to rapidly add obstacles to the C-space map while operating the robot. The purpose of the reflexive command filter is to protect the robot from erroneous higher-level commands that would result in collisions under high speed operation. The reflexive command filter is not concerned with fine motion planning or exact obstacle maps. For this reason it makes sense to ignore the degrees-of-freedom generated by the end effector in the C-space map. I implemented the reflexive command

filter primarily on the Robotics Research K-2107HR robot. For the case of the RRC robot this leads to a 4D C-space.

Currently, obstacles are defined either manually (by typing coordinates), "taught" by using the robot to touch points on an obstacle, or found using sonar-based world mapping. The sonar-based world mapping system represents new obstacles as sets of spheres. These spheres are added in to the C-space while the robot is investigating the workspace. Spheres, planes and lines are ideal primitives when using the template method.

Using the C-space generator, it is possible to generate C-space obstacles in a matter of seconds (or even faster depending on resolution level). In our current implementation, the finest resolution of configuration space is discretized into 64^4 "voxels" in 4-D, and these voxels are bit-mapped in RAM.

It should be noted that a database of templates must be precomputed off line. Since this process need only be performed once for each robot type, the computational efficiency of template generation is not crucial. Nonetheless, this step does require a large number of computations, and some efficiency improvement is desirable. For this purpose, I have included a conservative test for the possibility of collisions, based on geometric link and obstacle simplifications with sweep volumes. For poses in which a collision is easily proven to be impossible, the detailed point-by-point collision test computations are skipped. In this manner, we significantly speed up the template generation process, yet

preserve simplicity of the code for practical development and debugging.

In Section 4.1 the 4D C-space generator will be described in more detail. Section 4.2 gives an overview of which methods were used to generate the C-space templates and to generate static C-space.

4.1 The 4D C-space Generator

I constructed two versions of the 4D C-space generator, one for the first four joints of the RRC robot and one for the four joints of an Adept-II robot. For the RRC version I used points and spheres as features, and for the Adept-II version I used points, spheres and planes as features.

When the basic idea behind the template method is to be able to pick a precomputed C-space for a particular feature, this means that the template for the feature must be either precomputed with respect to all feature parameters or be possible to easily extract from other templates corresponding to the same template.

A point is defined by three parameters, namely the point location (X, Y, Z) with respect to the joint frame. The templates for points must therefore be precomputed for all values of the parameters (X, Y, Z). A sphere is defined by four parameters (X, Y, Z, R) where R is the radius of the sphere and (X, Y, Z) the location of the sphere center with respect to the joint frame. A sphere thus requires a larger database than a point. A plane is defined by three parameters,

namely the distance $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ to the plane along the plane normal. One can also define the plane parameters as $(\vec{X}, \vec{Y}, D)$ where $(\vec{X}, \vec{Y})$ are two of the components in the unit plane normal vector, and D is the distance to the plane. Lines are defined by four parameters etc.

It should be noted that one must always get rid of at least one feature dependency with the help of a simple translation. The shape of all templates remains constant if the feature is rotated with respect to the z axis of any revolute joint of a robot, or along any coordinate axis corresponding to a prismatic joint. If a feature is rotated θ degrees around the z axis of joint $\mathcal{J}$, the corresponding template will be purely translated θ degrees in C-space. This is evident from the definition of the DH-parameters. If the z axis of joint $\mathcal{J}$ remains stationary, which is usually true for the first joint in all non-mobile manipulators, this fact can be used to reduce the parameter dependency in the template generation process. It should also be noted that if the first three joints of a robot are prismatic, i.e. a cartesian robot, we can reduce the parameter redundancy by three parameters. I will refer to this as *feature parameter dependency reduction.* In general this means any C-space is purely translated if the corresponding work space obstacles are rotated around the z axis of the first rotary joint.

For the RRC version of the 4D C-space generator I used 2D-templates and for the Adept-II version I used 1D-templates. When using 1D-templates,

one must first find the parameter values of a feature with respect to the first joint coordinate frame, read the corresponding 1D-template, and place it in C-space. Secondly, for all discretizations of the first joint, the parameter values of a feature with respect to the second joint coordinate frame is found, the corresponding 1D-template is read, and placed in C-space. The same is done for the third and fourth joint. When 2D-templates are used, the parameter values of a feature with respect to the first joint coordinate frame are found, the corresponding 2D-template is read, and placed in C-space. Joint two and three are treated in a similar fashion. When using 2D-templates we do not have to do this procedure for joint four, which will save us computation time. However, in the case of the RRC, link-3 which is the elbow, is very small and can be considered a part of link-2 without great loss of accuracy, this procedure is done for joint-1 and joint-3 only. Using 1D-templates saves us memory when the 1D-templates usually are just one or two numbers, while the 2D-templates are 2D-shapes. On the other hand, the 2D-template method is considerably faster when the time consuming joint-4 loop is cut out. The 2D-template method requires circa ten times as much memory but is on the other hand circa ten times faster. The choice between the two should be made based on memory availability, speed requirements, accuracy requirements, and the dimension of the C-space.

4.2 Configuration Space Calculations

Using templates is a very fast way to generate C-space. However, the templates themselves must still be computed using "standard methods". Further, when using the template method all work space obstacles are approximated with a set of features like spheres, points, planes and lines. In most cases relating to collision avoidance and gross motion planning this is fine. However, in some cases where the workspace or parts of the workspace is expected to always remain the same, and high accuracy is desired, it might be desirable to use a "standard method" for C-space generation. Even though the C-space generation is less time critical in these cases, it should be noted that the C-space generation can be so time and resource consuming that it becomes a big problem, if not carried out with some efficiency.

When generating templates or static C-space for the RRC arm we cannot use the Minkowski set differences, due to the complexity of the RRC geometry and kinematics. The Jacobian-based method is not appropriate for this purpose either when it only generates local approximations. Further, the sweep volume method would in this case generate such complex volumes that the coding process would be very difficult.

It is easy to generate code using the point evaluation method. I used the point evaluation method to generate static C-space for the RRC. However, the point evaluation method is very time consuming run-time wise. When using

the point evaluation method I modeled the RRC robot as a set of cylinders, cones and spheres and checked algebraically for intersections at 128^4 discrete points in C-space. I discretized C-space into 64^4 "voxels" which means that each voxel contained $3^4 = 81$ inspection points. If any of these inspection points represented an intersection the voxel was considered taken. This is illustrated in Figure 4.2. When the C-space seldom contains thin or tiny objects, for real robots, even though the corresponding work space obstacles do, I considered 128^4 to be a sufficient approximation. Further, the problem resulting from the thin and tiny objects that appear in C-space are easily eliminated by conservative work space obstacle and robot approximations.

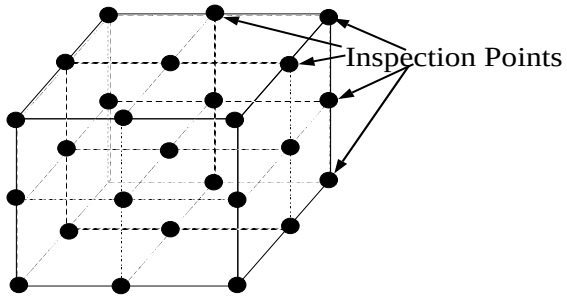


Figure 4.2: If one inspection point is taken this entire voxel will be considering taken. If the inspection point also belongs to a neighboring voxel, that voxel will also be considered taken

I also used the boundary equation method to generate templates for the RRC robot. The problem with the boundary equation method is that it results in very complex equations which in general are not analytically solvable. It was very difficult and time consuming to derive the boundary equations, and the run time for the resulting C-space generation programs were disappointing.

The reason for the disappointing run time in this case was that the numerical methods I used to solve the resulting "highly unstable" and very complex equations were inefficient. In fact the point evaluation method was a good candidate as an alternative "numerical method" for solving these equations.

However, I was successful in using a boundary equation method for generating 2D-point-templates for the RRC robot. I will refer to this boundary equation method as the "cut method". In the cut method I express the cones which forms the links of the RRC with the following equation : $y = \sqrt{(R_0 - kz)^2 - z^2 - x^2}$ for $Z^{top} \geq z \leq Z^{bott}$. This equation describes the cone surface in the coordinate frame corresponding to the link the cone is a part of. The cone surface is uniquely defined in this coordinate frame. In the case described here the point is defined in the base frame and the cone in the joint-2 coordinate system. I then form the equation $\vec{x_p^0} = A_2^0 \vec{x_c^2}$, where $\vec{x_p^0} = (x_p, y_p, z_p)$ is the known point location with respect to the base frame and $\vec{x_c^2} = (x, \sqrt{(R_0 - kz)^2 - z^2 - x^2}, z)$ for $Z^{top} \geq z \leq Z^{bott}$ is the cone equation in frame two. A_2^0 is the transformation matrix from joint-2 to the base frame. The cone surface equation is essentially translated from frame-2 to the base frame and equated with the point for which we wish to compute the corresponding C-space (template).

This equation contains four unknowns $(x, z, \theta_1, \theta_2)$ and three equations. The reason for this is that point will intersect with the cone surface along

an elliptical path I will refer to as the *cut*. The cut is now found by keeping one of x or z constant and solving for θ_1 and θ_2. It must be predetermined which one of x and z should be kept constant in different expected results of θ_1 and θ_2 for the sake of equation stability. I succeeded in solving this equation analytically, and by varying one of x or z I found the θ_1 and θ_2 that corresponded to the point-cone surface intersections. Attempts to apply this method to higher dimensions or planes and lines resulted in equations which were not analytically solvable. Another implementation using boundary equations is described in [33]. In this paper the robot links are assumed to be sticks, and the world space obstacles polyhedral obstacles with surfaces consisting of triangular patches.

Another approach to computing templates and static C-space which I found more fruitful was what I will refer to as the "layered intersection predetermination method". In the *layered intersection predetermination method* the possibility for obstacle-robot intersection is determined in three steps.

1. Sweep volumes, or the needle point method, in combination with simple and conservative approximations of the robot links and obstacles, are used to determine the possibility of robot link and obstacle intersection. Smaller sweeps are used to narrow in on possible intersections. Figure 4.3 illustrates sweep volumes derived from approximations.

2. If an intersection is possible, use the point method in combination with

simple but conservative approximations of obstacles and robot links to check whether an intersections is possible, and if so guaranteed. This is illustrated in Figure 4.4.

3. If an intersection is possible but not guaranteed, perform a complete intersection check.

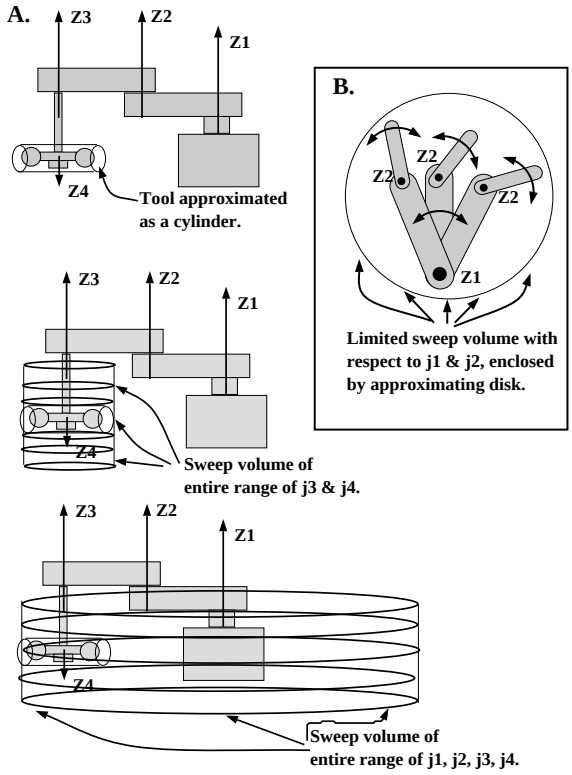


Figure 4.3: A. Demonstrates full range sweeps of a cylinder approximation of the tool. B. Demonstrates an approximation of partial range sweeps.

In all three steps intersection determination procedures for different sets of obstacles are used. Examples of such procedures are,

1. To check for the intersection of two spheres of radius R_1 and R_2, determine

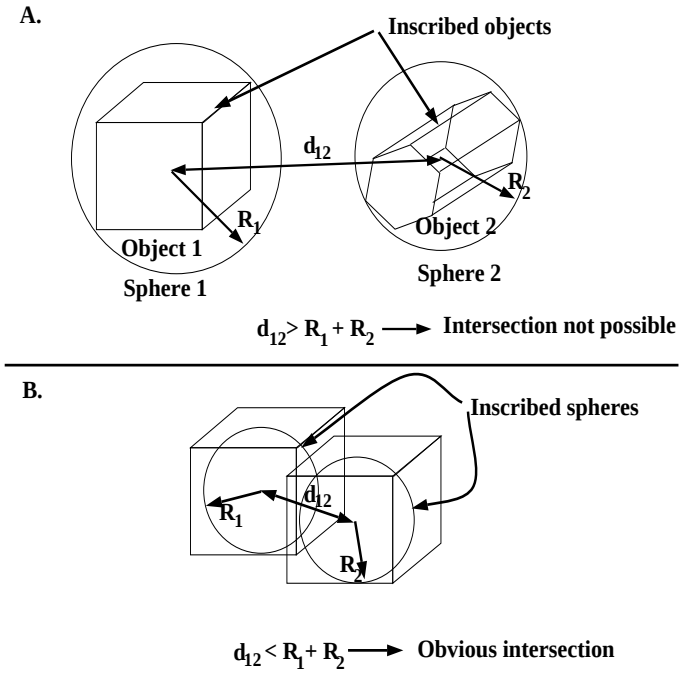


Figure 4.4: A. The possibility of intersection between two objects is determined. B. Using inscribed spheres it is easy to show intersection.

if the distance d between the two spheres is less than $R_1 + R_2$.

2. Check for intersection between a cylinder and a sphere. First compute the shortest distance (normal distance) between the sphere center and the cylinder-center, and the corresponding normal intersection point on the cylinder-center. This is a simple operation if the equation for the cylinder-center line is known. If the distance is larger than the combined radiuses of the cylinder and the sphere we cannot have an intersection. We have an intersection, if the normal distance is less or equal to the combined radiuses of the cylinder and the sphere, and the corresponding normal intersection point is inside the cylinder, or the hypotenuse formed

by, the offset line from the cone-end and the normal intersection point, and the normal minus the cylinder radius, is larger than the sphere radius.

3. Check for intersection between a "cut-cone" (cone with its top cut-off) and a sphere. First compute the shortest distance between the sphere center and the cone-center, just like in the case above. This check is in the continuation similar to the one above, except that, the equations contain more complex trigonometry due to the varying radius of the cone, and the fact that the cone-center normal will not be normal to the cone surface.

4. To check for the intersection of two convex polyhedra described as the intersection of a number of planes, $A_i^1 x + B_i^1 y + C_i^1 z + D_i^1$ (obstacle one), $A_j^2 x + B_j^2 y + C_j^2 z + D_j^2$ (obstacle two), where the normals of the planes are pointing inwards to the obstacle, do the following. Points inside or on, for example, the obstacle are characterized by, $\forall i \; A_i^1 x + B_i^1 y + C_i^1 z + D_i^1 \geq 0$. For intersection we have that either, all corners of one of the obstacles are inside the other obstacle, or at least one edge of obstacle one intersects one surface of obstacle two. It also true that if, one corner of obstacle one is inside obstacle two, or one corner of obstacle two is inside obstacle one we for sure have an intersection. If we suspect an intersection we first

compute the corners by solving,

$$\begin{aligned} A_i^n x + B_i^n y + C_i^n z + D_i^n &= 0 \\ A_j^n x + B_j^n y + C_j^n z + D_j^n &= 0 \\ A_k^n x + B_k^n y + C_k^n z + D_k^n &= 0 \end{aligned}$$

for all combinations of (i, j, k) for both objects. However, this equation might yield some "false" corners. These false corners could be removed by making sure that $A_i^n x + B_i^n y + C_i^n z + D_i^n \geq 0$ for all planes which forms the convex polyhedra. Now, check if the any of the generated corner points for obstacle two fulfill the $\forall i \ A_i^1 x + B_i^1 y + C_i^1 z + D_i^1 \geq 0$ requirement, and vice versa for obstacle one. If this is not true we check if any of the edges intersect any of the surfaces of the other obstacle. First we need to find the equation of the edges which can be done by for example generating the equations for the lines between all points, and check if they intersect two planes, if so the lines corresponds to edges. Secondly, check if any of these edges intersect any of the planes for the other obstacle, at points which are inside the polygon formed by the corresponding corner points. It should be noted that before this computation is done, simpler checks should have been done like, checking the intersection between the enclosing spheres (cylinders). If the obstacles are not convex the obstacles must be subdivided into convex parts for this algorithm to work.

5. To check for the intersection of two arbitrary polyhedra described as a set of corner points and edge-line equations, do the following. For intersection we have that either, all corners of one of the obstacles are inside the other obstacle, or at least one edge of obstacle one intersects one surface of obstacle two. It also true that if, one corner of obstacle one is inside obstacle two, or one corner of obstacle two is inside obstacle one we for sure have an intersection. Find the surface equations by using three adjacent corner points to solve for A, B, and C in the following equation system.

$$\begin{aligned} Ax_1 + By_1 + Cz_1 + D &= 0 \\ Ax_2 + By_2 + Cz_2 + D &= 0 \\ Ax_3 + By_3 + Cz_3 + D &= 0 \end{aligned}$$

where D is picked arbitrarily. This equation can now be normalized, and the direction of the normal can be chosen so that an arbitrary point inside the obstacle, or if the obstacle is convex, another corner point, would give a positive result if inserted in the plane equation. We first need to check if any obstacles cornerpoint is inside any of the other obstacles convex subdivisions. If this is not true we check if any of the given edge equations intersect any of the planes corresponding to the other obstacle at a point which is inside the polygon formed by the corresponding corner points.

The types of objects I used to approximate robot links and work space obstacles with were spheres and cylinders. The primitives I used to describe my actual obstacles and robot links with were, cylinders, cones, spheres, and convex polyhedra. This method was primarily used to generate the static C-space for the Adept-II robot, but also to a limited extent used to generate the static C-space for the RRC robot.

4.3 Conclusions and Overview for Chapter 4

The configuration space of a robot $\mathcal{A}$ is the space $\mathcal{C}$ of all configurations of $\mathcal{A}$. There are seven standard methods for generating configuration space maps, point evaluation, the boundary equation method, Minkowski set differences, the needle method, the sweep volume method, the Jacobian based method, and the template method. The template method is particularly useful for quick configuration space generation.

This chapter described a configuration space generator, which generated configuration space obstacles very quickly utilizing the template method. Using the C-space generator, it is possible to generate C-space obstacles in a matter of seconds (or even faster depending on resolution level).

Using templates is a very fast way to generate C-space. However, the templates themselves must still be computed using "standard methods". Further, when using the template method all work space obstacles are approximated

with a set of features like spheres, points, planes and lines. In most cases relating to collision avoidance and gross motion planning this is fine. However, in some cases where the workspace or parts of the workspace is expected to always remain the same, and high accuracy is desired, it might be desirable use an exact computation method for C-space generation. The layered intersection predetermination method was an exact computation method introduced in this chapter.

The reflexive command filter to be described in Section 5 relies on a predefined obstacle map which in our case is a configuration space map. The use of configuration space in the context of reflex control, or for any other purpose, have a number of advantages and disadvantages. The advantages are:

1. The robot is represented as a point in configuration space. It easy to create algorithms for navigation or obstacle avoidance of a point.

2. Dynamic relationships are often easy to express in terms of the configuration space, particularly if the configuration space is the robot joint space. This is particularly useful in the context of reflexive obstacle avoidance.

3. In configuration space each pose of the robot maps into a single point. The configuration space does not contain any singularities, multiple solutions, or redundancies.

The disadvantages are:

1. The dimension of the configuration space is the number of the parameters representing a configuration. This means that highly redundant robots result in very high-dimensional configuration spaces. This makes discretized, or even descriptive configuration space maps of highly redundant robots very hard to store in RAM.

2. For complex robot systems it is very hard to compute configuration space obstacles, which makes the configuration space approach very time consuming. The template method partially solves this problem.

For system of high dimensionality it may not be feasible to pre-compute a full configuration space map. In this case the configurations space must be computed locally. However, it should be noted that on-line obstacle intersection detection in work-space and local configuration space map generation are very similar processes. An algorithm which locally computes all possible collisons is in a sense computing local configuration space obstacles. In this context the template method for generating configuration space obstacles could be very useful.

Chapter 5

Reflexive Collision Avoidance for System Preservation and Reliability

A promising approach to on-line automatic obstacle avoidance is the use of artificial potential functions. Potential functions are usually constructed about obstacles in task space, and gradients of these functions define virtual repulsive forces on a system to achieve collision avoidance. In most cases [2, 36, 41, 38], either these approaches do not consider actuator effort saturation, and "guarantee" collision avoidance only through the assumption of unlimited force and torque availability, or they invoke extremely conservative approximations. Further, these methods often focus on collision avoidance with respect to the end-effector only, not the entire arm. To include protection of points on the arm as well, and to consider complex workspace, the computational requirements of these methods can become prohibitive, leading to either very slow motion or loss of collision protection.

Reflex control for obstacle avoidance is similar to potential functions in theory. However, the reflex controller exhibits the following virtues: It does

not suffer from unrealistic or overly restrictive assumptions on robot dynamics; its complexity does not increase with higher dimensions; it does not fail or slow down as environment complexity increases; it does guarantee collision avoidance (at least for static obstacles) and can be computed on line; it does not suffer from excessive influence of obstacle repulsion fields; its complexity increases only linearly with higher dimensions; it is ideally transparent, i.e., it does not introduce any significant distortion of higher-level controls in all but emergency cases; it fits in control hierarchies, and it can be used for both system preservation and as a building block in efficient path planning algorithms.

Reflex control for obstacle avoidance was introduced in [56], and further developed and applied in different applications in [55, 9, 58, 76, 77]. The reflex controller approves or disapproves higher-level commands, based on a rapid evaluation of a neighborhood of the robot's environment. This way, the reflex controller acts like a command filter that ideally is transparent, i.e., does not introduce any significant distortion of higher-level controls in all but emergency instances. For this reason our reflex controller will be referred to as the reflexive command filter in the remainder of this thesis. This quality makes the reflexive command filter amenable to control hierarchies and makes it possible to use it for both system preservation and as a building block in efficient path planning algorithms.

This chapter describes a generalized, multifunctional and flexible method to implement the reflexive command filter. Other types of implementations are described in [55, 9, 58, 76, 77]. The reflexive command filter described here can theoretically be inserted at any level in a control hierarchy and generate safe commands to underlying levels. The commands generated to the underlying levels are filtered with respect to collision avoidance, using a map of the immediate configuration space and sufficient knowledge about underlying levels (including the dynamics of the robot). An overview of the reflexive command filter is given in Figure 5.1.

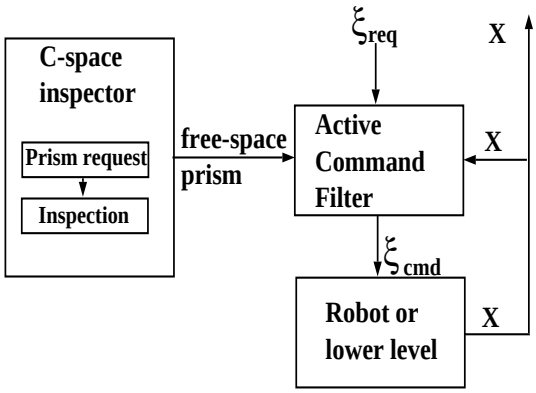


Figure 5.1: The reflexive command filter.

Our reflexive command filter consists of two sub-modules which operate asynchronously. One module is the *C-space inspector* and the other is the *active command filter*. The C-space inspector corresponds to the virtual sensor of the reflexive command filter and the active command filter is the active component of the reflexive command filter (For a review of "configuration

space" or "C-space", see [11, 59, 46, 45]). The C-space inspector is given a request for a prism in C-space[1], which it inspects and approves if the request is obstacle-free. If the prism is not obstacle-free, the C-space inspector will return an obstacle-free subset of the request. The obstacle-free subset of the request returned by the C-space inspector becomes the *approved free-space prism*. The requested prism can be generated externally by a higher-level or internally by the reflexive command filter. Section 5.3 describes how to generate an appropriate prism request.

The active command filter receives a generalized request $\boldsymbol{\xi}_{req}$ which can be a position, velocity, acceleration, or torque request from a higher-level, and the active command filter issues a corresponding generalized command $\boldsymbol{\xi}_{cmd}$ directly to the robot or to other underlying levels. The command issued, $\boldsymbol{\xi}_{cmd}$, must be safe with respect to collision avoidance, where "safe" will be defined formally in Section 5.2. For all requests which would not result in the robot's leaving the approved free-space prism, the active command filter controller approves the request as an identical command. If, however, the request would cause the robot to commit to a dangerous or fatal future, the reflexve command filter instead issues an alternative command based on a predefined *braking policy*.

Since the reflexive command filter can easily be incorporated with other

[1]All prisms referred to here are aligned with the C-space axes and have dimension N, the dimension of the C-space.

control schemes with minimum non-essential influence on higher-level controls, it serves as a natural building block in more advanced obstacle avoidance and path planning schemes. In Section 7 we describe a geometric on-line 4-D path planning algorithm that uses the reflexive command filter as a building block.

5.1 The Braking Policy and Braking Prism

A crucial construct of our reflexive collision avoidance scheme is the concept of the *braking prism*. The braking prism is an N-dimensional prism in C-space, aligned with the C-space axes, which is known to contain a robot's trajectory resulting from enforcement of a *braking policy*. A braking policy is a prescription for generating commands to the robot (or underlying levels) as a function of robot state, $\mathbf{x}$, that brings all axes of the robot to a halt. It is not necessary that the policy prescribe immediate deceleration of any of the axes. Rather, it is only required that the trajectory resulting from application of the braking policy is predictable, and that this trajectory (if carried to completion) would terminate in the robot at rest. If the braking trajectory under a given braking policy is predictable, then we can define a corresponding *braking prism* in C-space which contains the braking trajectory.

It will be useful to define a compact notation to describe prisms in N-space. In N-space, there will be 2^N vertices of any rectangular (hyper-) prism. We will consider rectangular prisms which are aligned with a reference coordinate

frame, i.e., with prism edges parallel to coordinate frame axes. Among such prisms, we can define a "most negative" (least positive) vertex and a "most positive" (least negative) vertex. More formally, if v_i^j is the ith coordinate of the jth vertex, we may define the scalar quantity $d_j = \sum_{i=1}^{N} v_i^j$. The vertex corresponding to the minimum value of d_j is labeled "$\mathbf{v}^{\min}$"(the most negative vertex), and the vertex corresponding to maximum d_j is labeled "$\mathbf{v}^{\max}$" (the most positive vertex). These vertices are uniquely determined[2]. Note that these two opposite vertices $\mathbf{v}^{\min}$ and $\mathbf{v}^{\max}$ completely specify the geometry of any aligned, rectangular prism in N-space. We will invoke these definitions for all C-space prisms considered in this presentation.

In particular, we may consider braking prisms, which contain braking trajectories. Application of a braking policy beginning at time t results in a trajectory from an initial state specified by $\mathbf{q}(t), \dot{\mathbf{q}}(t)$ (joint-space pose and joint velocities) to a final rest state $\dot{\mathbf{q}} = \mathbf{0}$ at pose $\mathbf{q}_f$. Any such braking trajectory can be bounded by an aligned, rectangular prism in N-dimensional joint space. Among all such bounding prisms, there is a unique prism of minimum volume. We shall define this min-volume prism to be the *braking prism* with respect to the specified braking policy and initial state, as illustrated in Figure 5.2.

The braking prism can be described in terms of its opposite vertices, $\mathbf{B}^{\min}$

[2]Prisms aligned with the reference coordinate frame.

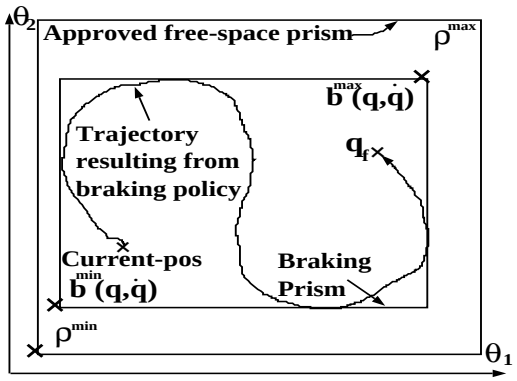


Figure 5.2: Trajectory resulting from a braking policy defines a braking prism.

and $\mathfrak{b}^{\max}$. Since the braking prism is uniquely defined by a braking policy and by initial conditions (the state of the robot at time t), we can express the bounding vertices of the braking prism as a function of state: $\mathfrak{b}^{\min}(\mathbf{q}(t), \dot{\mathbf{q}}(t))$ and $\mathfrak{b}^{\max}(\mathbf{q}(t), \dot{\mathbf{q}}(t))$, or more compactly $\mathfrak{b}^{\min}(\mathbf{q}, \dot{\mathbf{q}})$ and $\mathfrak{b}^{\max}(\mathbf{q}, \dot{\mathbf{q}})$, or $\mathfrak{b}^{\min}(t)$ and $\mathfrak{b}^{\max}(t)$.

It is important to recognize that $\mathfrak{b}^{\min}$ and $\mathfrak{b}^{\max}$ are defined as a function of state, regardless of whether or not the associated braking policy is applied. These vertices constitute a compact description of the predicted behavior of the robot under the presumption of initiated braking. If the respective braking policy *is* invoked (and continued) beginning at time t_o, then all $\mathfrak{b}^{\min}(t)$ and $\mathfrak{b}^{\max}(t)$ will remain confined within the prism defined by $\mathfrak{b}^{\min}(t_o), \mathfrak{b}^{\max}(t_o)$ for all time $t > t_o$. More strongly, $\mathfrak{b}^{\min}(t)$ and $\mathfrak{b}^{\max}(t)$ will converge monotonically on the final pose, $\mathbf{q}_f$. Alternatively, if the braking policy is not imposed at time t_o, then the corresponding braking-prism vertices $\mathfrak{b}^{\min}(t)$ and $\mathfrak{b}^{\max}(t)$ will

have trajectories which may well leave the prism defined by $\mathfrak{b}^{\min}(t_o), \mathfrak{b}^{\max}(t_o)$.

Ordinarily, the braking-prism vertices $\mathfrak{b}^{\min}(\mathbf{q}, \dot{\mathbf{q}})$ and $\mathfrak{b}^{\max}(\mathbf{q}, \dot{\mathbf{q}})$ will be continuous functions of state. For ease of presentation, we will restrict our consideration to those braking policies for which $\mathbf{b}(\mathbf{q}, \dot{\mathbf{q}})$ are differentiable in $\mathbf{q}$ and $\dot{\mathbf{q}}$. In the cases where $\mathbf{b}(\mathbf{q}, \dot{\mathbf{q}})$ is not differentiable it might still be possible to find an estimation of $\Delta b(t)$.

Under the condition of differentiable $\mathbf{b}(\mathbf{q}, \dot{\mathbf{q}})$, (where $\mathbf{b}$ represents a braking prism vertex) we can predict the trajectory of $\mathbf{b}(\mathbf{t})$ by differentiation:

$$d\mathbf{b}/dt = (\partial \mathbf{b}/\partial \mathbf{q}) d\mathbf{q}/dt + (\partial \mathbf{b})/\partial \dot{\mathbf{q}}) d\dot{\mathbf{q}}/dt$$

In the above, the term $d\dot{\mathbf{q}}/dt$ depends on the dynamic equations of the robot and on the control command to be exerted. Thus, we can evaluate $\dot{\mathfrak{b}}^{\min}$ and $\dot{\mathfrak{b}}^{\max}$, the evolution of the braking prism, as a function of state and a generalized request, $\boldsymbol{\xi}_{\text{eq}}$.

In implementation, we will perform evaluations at discrete time intervals of Δt, resulting in predicted increments of $\mathbf{b}$, $\Delta \mathbf{b}$, rather than continuous time derivatives. In these cases, we will presume a discretized equivalent restriction on differentiability. Specifically, it will be assumed that the trajectory of a braking-prism vertex $\mathbf{b}$ moving from $\mathbf{b}(t)$ to $\mathbf{b}(t + \Delta t) = \mathbf{b}(t) + \Delta \mathbf{b}$ is entirely contained within the prism defined by $\mathbf{b}(t)$ to $\mathbf{b}(t) + \Delta \mathbf{b}$, regardless of the command imposed.

With the definitions above, we have an efficient means by which to test

impending danger of collision. Should it be necessary to stop the robot from colliding with a known obstacle, we can rapidly detect whether the collision can be prevented by applying some specified braking policy. Further, we have a means of predicting the onset of danger, by invoking computations of the time derivative of the braking-prism vertices. We can use these properties in the construction of a general reflexive collision-avoidance scheme, which we describe in the following section.

5.2 The Active Command Filter

The active command filter continuously makes rapid decisions whether to approve a request as an identical command, or to issue instead a command derived from a braking policy. To make its decision, the active command filter continuously evaluates the braking prism corresponding to a default braking policy and to the specified request as a function of robot state. Multiple alternative braking prisms corresponding to alternative braking policies may be considered. As long as at least one braking prism would not exit the approved free-space during time Δt (using the input request), that request is passed through as an approved command. If, however, the request would cause all braking prisms to exit the approved free-space volume, the request is denied, and a valid braking policy is immediately invoked instead. It should be noted that, by invoking a valid braking policy, we guarantee that the robot will stay

within the approved free-space prism and that this braking policy will remain valid. Examples demonstrating how to define braking policies and compute the corresponding braking prisms are given in Section 5.4.

We will define a state of the robot as *projected safe* with respect to an input request, a specific braking policy, and a specified free-space prism as follows. If the braking prism can be expected to remain within the free-space prism in the immediate future (i.e., one time step Δt) assuming execution of the request as a command, then that state is *projected safe*. This is the condition which the active command filter must continuously evaluate to approve or deny input requests. A more detailed algorithm describing this process is given below.

A braking prism defined by points $\mathbf{b}^{\min}(\mathbf{q}, \dot{\mathbf{q}})$ and $\mathbf{b}^{\max}(\mathbf{q}, \dot{\mathbf{q}})$ is illustrated in Figure 5.2. Further, the approved free-space volume is defined by opposite prism vertices $\boldsymbol{\rho}^{\min}$ and $\boldsymbol{\rho}^{\max}$. *Projected safe* can be defined as follows:

The braking policy is projected safe if for each joint coordinate i the corresponding braking prism fulfills the following requirement.

$$b_i^{\min}(q, \dot{q}) + \Delta b_i^{\min} \quad > \quad \rho_i^{\min}$$

and

$$b_i^{\max}(q, \dot{q}) + \Delta b_i^{\max} \quad < \quad \rho_i^{\max}$$

Where $\boldsymbol{\rho}^{\min}$ and $\boldsymbol{\rho}^{\max}$ are defined for the free-space prism as $\mathbf{b}^{\min}$ and $\mathbf{b}^{\max}$ are defined for the braking prism. $\Delta\mathbf{b}^{\min}$ and $\Delta\mathbf{b}^{\max}$ are the expected changes

of $\mathfrak{b}^{min}$ and $\mathfrak{b}^{max}$ during the next cycle of the active command filter, assuming the request were to be approved. If Δt is small and the braking policies closely corresponds to maximum braking, the reflexive command filter will be transparent in the sense that higher-level commands will be replaced only if necessary.

Heuristically, this mathematical definition means that the braking policy is *projected safe* if the corresponding braking prism is contained in the approved free-space prism and its boundaries are either moving away from the boundaries of the approved free-space prism, or the expansion of the braking prism is small enough not to exceed the boundaries of the approved free-space prism during the next cycle.

The operation of the active command filter can be summarized as follows :

- The active command filter computes a set of braking prisms corresponding to a set of braking policies and determines whether any are projected safe.
- If at least one of the braking policies is projected safe, the current command request is approved verbatim as a command. If none of the braking policies are projected safe with respect to the current command request, the request is denied. In that case, a command is generated corresponding to a braking policy for which the braking prism is currently contained within the free-space prism.

- When the C-space inspector generates a new free-space prism, the active command filter attempts to restrict the braking prism or prisms to the intersection of the new free-space prism and the previously approved free-space prism. The new free-space prism will not be approved until this is achieved. The motivation behind this action is to guarantee that the braking policy at all times is projected safe, even at the instances when the approved free-space prism is updated. This is illustrated in Figure 5.3.

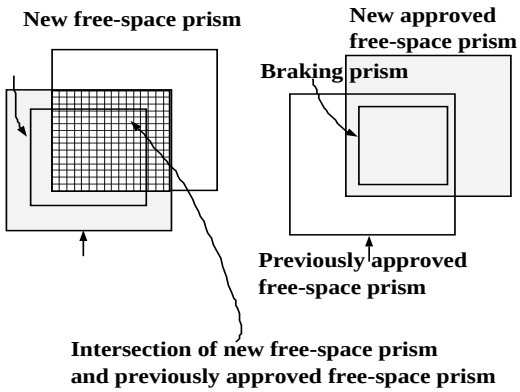


Figure 5.3: The new free-space prism is not approved until the braking prism is contained within the intersection of the new free-space prism and the previously approved free-space prism. The approved free-space prism is displayed in gray.

The active command filter guarantees that the braking prism is always contained in the approved free-space prism, even when the free-space prisms are updated. By definition the robot will always be contained in the current braking prism. Since approved free-space prisms are obstacle-free, the reflexive command filter thus guarantees collision avoidance.

The execution time of the active command filter increases linearly with the dimension of the C-space. In our 4-D implementation, the active command filter runs at a constant cycle rate of 1900Hz.

5.3 The C-space Inspector

Our second important module is the C-space inspector, which evaluates whether a proposed prism in C-space is obstacle-free. The basic function of the C-space inspector is illustrated in Figure 5.4. The C-space inspector is given a request for a prism. In this case, the requested prism contains obstacles and will not be the returned free-space prism. Instead, a subset of the requested prism that is free from obstacles is returned. The free-space prism is generated by inspecting the difference between the last-approved prism and the new request. In other words only uninspected C-space is inspected.

There is more than one way to choose an obstacle-free subset of the requested prism if the requested prism contains obstacles. It is desirable to choose an obstacle-free subset that minimizes the influence of the obstacle. To achieve this, the free-space prism is chosen to accommodate future motion of the braking prism.

In our case, the uninspected C-space is incrementally inspected by inspecting segments of C-space, called search-fronts. The search begins at the borders of the last approved free-space prism, and is expanded into the new requested

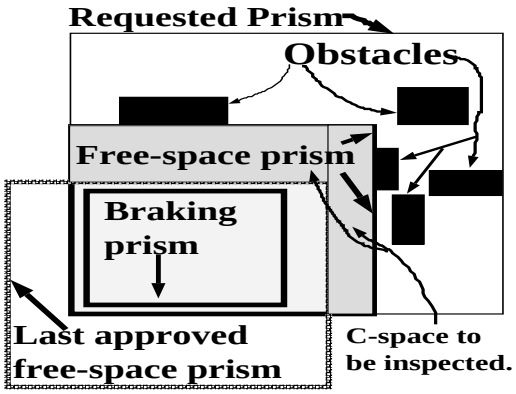


Figure 5.4: A new free-space prism in a cluttered environment. When the braking prism is contained within the new free-space prism it becomes the new approved free-space prism. The C-space that had to be inspected is displayed in dark gray.

prism in a wave-like manner, as illustrated in Figure 5.5. When an obstacle is encountered at a search-front, the expansion is halted in that direction. In Figure 5.5, the search-front expanding in the θ_1 direction encounters an obstacle, and further expansion in the θ_1 direction is halted. To account for future expansions of the braking prism we assign different speeds to the search fronts. The speeds of the search fronts depend on the expected future motion of the braking prism. In Figure 5.6 the braking prism is moving rapidly in the θ_1 direction, which results in a higher speed of the search front expanding in the θ_1 direction. Consequently, the obstacle in Figure 5.6 causes an expansion halt in the θ_2 direction.

The resulting free-space prism is displayed in gray in Figure 5.6. For the reasons explained in Section 5.2, the new free-space prism will not be approved until the braking prism is contained within the new free-space prism. In Figure

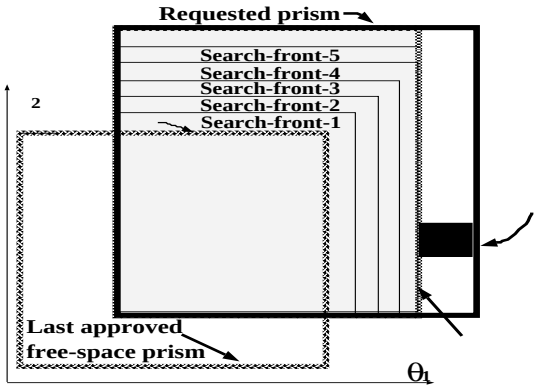


Figure 5.5: Search-fronts originating from last approved free-space prism.

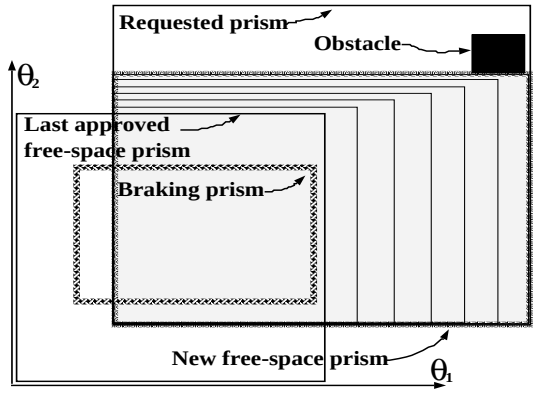


Figure 5.6: The dynamics of the robot defines speed of search-fronts.

5.6 the braking prism is still not contained in the new free-space prism, and for that reason the new free-space prism is still not approved.

The C-space inspector is *fail-safe* in the sense that if it would halt unexpectedly, the last free-space prism would still remain a valid, safe constraint. The active command filter, on the other hand, is, in general, not fail-safe. The execution time of the C-space inspector grows geometrically with the dimensionality of the C-space. The execution time of the C-space inspector further depends on the size of the region to be inspected. Thus, efficiency is improved

when only local regions of C-space are inspected.

If the reflexive command filter does not receive prism requests from higher levels, the reflexive command filter generates its own prism requests (based on the generalized requests it receives from higher-levels). This can be done by C-space inspector or a second separate module. It is not crucial for obstacle avoidance how this is done. However, to minimize the influence of the reflexive command filter, these prism requests should conservatively contain the current braking prism, but still be small enough not to cause excessive C-space inspection time.

We have found acceptably fast performance in our 4-D implementation. We discretized C-space into 64^4 "voxels." In our 4-D implementation, we have a worst-case execution time of 10ms for an inspection of a prism consisting of 5^4 or 625 voxels. At this size and update rate, our implementation of the reflexive command filter did not noticeably affect the behavior of the robot in free space. An illustration of how the free-space prism evolves while the robot is moving among obstacles is given in Figure 5.7.

5.4 Examples of Braking Policies

We have implemented and experimentally verified a few braking policies. The braking policies we have used are all based on some type of non-overshooting servo-control. In these cases, the braking prism is particularly easy to compute.

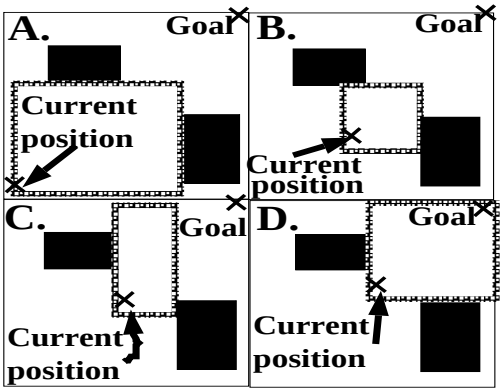


Figure 5.7: Free-space prism evolves in C-space while the robot is moving. One of its vertices is located at the current position and the opposite vertex at the point where the robot is expected to come a complete stop when applying the braking policy. Below we describe three different braking policies operating in three different settings. The settings are shown in Figure 5.8.

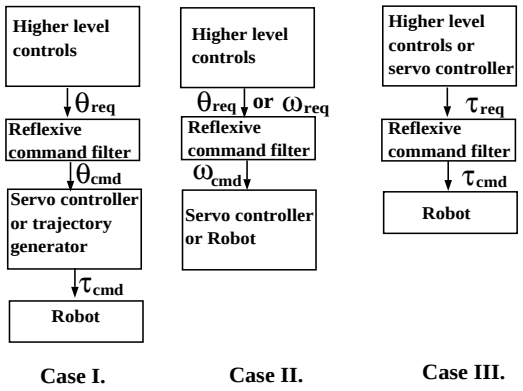


Figure 5.8: Three different settings for the reflexive command filter.

- Case 1 : The commands received by the reflexive command filter are position commands and the commands generated by the reflexive command filter to an underlying non-overshooting servo-controller, or trajectory generator, are position commands. In this case the braking policy

is particularly simple. All position requests located inside the approved free-space prism are approved, all others are replaced by the position corresponding to the intersection between the border of the free-space prism and the line-segment from the current position to the requested position. This is illustrated in Figure 5.9. All position commands generated to the underlying levels are inside the approved free-space prism, and the non-overshooting servo-controller or trajectory controller guarantees that the robot does not overshoot these positions. For this reason the reflexive command filter guarantees obstacle avoidance. It should be noted that if any of the reflex control modules would fail, the last approved command would still remain valid, and we are still guaranteed obstacle avoidance. For this reason the reflexive command filter in this setting is said to be fail-safe. This approach to reflexive obstacle avoidance has been tested on three different manipulators, a high-speed 2-link planar arm [56, 54], a General Electric GP-132 industrial robot [58, 11, 9], and on the Robotics Research Corporation K-2107HR robot arm [76, 77].

- Case 2 : The commands received by the reflexive command filter are velocity or position commands, and the reflexive command filter generates approved velocity commands to a servo-controller or to a robot emulating velocity control at the analog level. In this case the braking policy we are

using is $\omega_{cmd_i} = \mathbf{sgn}(d_i)\sqrt{2|d_i|\alpha_{max_i}}$, where α_{max_i} is the maximum acceleration for joint i (which is chosen to achieve close to maximum braking without overshoot), $|d_i|$ is the joint distance to where we want to come to a complete stop, and ω_{cmd_i} is the velocity command. If this braking policy is used, $|d_i|$ would correspond to the distance to one of the vertices of the approved free-space prism. The braking prism in this case will have one vertex at the current position and the opposite vertex at the *braking point* $d_{brk_i} = \omega_i^2/(2\alpha_{max_i})$. If the braking policy is projected safe, the velocity request is approved, and if the request is a position request, maximum velocity is commanded. If the braking policy is not projected safe, the request is replaced by $\omega_{cmd_i} = \mathbf{sgn}(d_i)\sqrt{2|d_i|\alpha_{max_i}}$. The braking policies described here and in the next case are derived from the reactive trajectory generator described in [77].

- Case 3 : The commands received by the reflexive command filter are torque commands and the commands generated by the reflexive command filter directly to the robot are torque commands. Our braking policy in this case is $\tau_i = K_{p_i}(\theta_{des_i} - \theta_i) + K_{v_i}(\omega_{des_i} - \omega_i) + K_{a_i}\alpha_{des_i}$, where

$$\begin{aligned} \alpha_{des_i} &= -\mathbf{sgn}(d_i)\alpha_{max_i} \\ \omega_{des_i} &= \mathbf{sgn}(d_i)\sqrt{2|d_i|\alpha_{max_i}} \\ \theta_{des_i} &= \theta_i + \omega_i\Delta t - \mathbf{sgn}(d_i)\alpha_{max_i}\Delta t^2/2 \end{aligned}$$

Where $\alpha_{\max_i}$ is the maximum acceleration, ω_i the velocity, d_i the distance to where we want to come to stop, and Δt is the loop time (time elapsing between command updates) of the active command filter. $\alpha_{\max_i}$, K_{p_i} and K_{v_i} are adjusted to achieve close to maximum braking without overshoot. The braking prism in this case will have one vertex at the current position and the opposite vertex at the braking point. If the braking policy is projected safe, all torque commands are approved. Otherwise the torque commands are issued as per the above braking policy. Another approach emulating direct torque control is described in [54]; experiments using this approach are described in [9, 59].

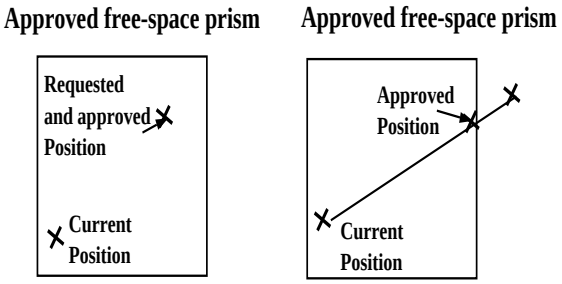


Figure 5.9: All position requests located inside the approved free-space prism are approved, all others are replaced.

It should be noted that it is not always possible to construct a non-overshooting braking policy without considerably affecting the speed and acceleration capabilities of the robot. This is, for example, the case when there is strong dynamic coupling among the links of the robot. In such cases, our braking policies may be overly conservative to guarantee no overshoot. Instead, braking policies

that allow overshoot should then be constructed. It is always possible to construct a braking policy provided the resulting braking trajectory is predictable enough that it can be bounded by a prism. The non-overshooting braking policies we have implemented are emulations of non-overshooting servo-controllers.

By a non-overshooting servo-controller we mean a servo-controller, which for all position commands, generates paths which monotonically converge towards the desired joint positions, if given from a zero initial velocity. Examples of such non-overshooting servo-controllers are soft (over- or critically-damped) PD-controllers, bang-bang, and sliding-mode-controllers. We have also developed and implemented what we call a reactive trapezoidal trajectory generator. The reactive trapezoidal trajectory generator generates (in a reactive manner) non-overshooting, nearly time-optimal trapezoidal velocity profiles to an underlying servo-controller. The reactive trapezoidal trajectory generator will be further described in 6.

5.5 Using the Reflexive Command Filter for Fault Tolerance and Autonomy

Fault tolerance and system reliability with respect to hardware, mechanical and sensor failures have been addressed by numerous scientists. An extensive overview of these issues, and failure rates for different components are given in

[27]. In [50, 49] the degradation of dexterity due to a joint failure in kinematically redundant manipulators is analyzed, and a failure tolerance measurement with respect to local dexterity is defined and used to find optimal fault tolerant configurations. In [44] a failure tolerance measurement is used to optimize the dynamic performance of redundant robots in the presence of joint failures. An expert system for fault detection and fault tolerance is described in [48]. This expert system is able to detect faults and use recovery data to reveal functional replacements for faulty components and indicate possible causes or effects of the failures. In [80] Wu, Hwang and Chladek presents an analysis of a dual-motor fault tolerant joint design for the space shuttle remote manipulator system.

The primary objective of the reflexive command filter is to guarantee system survivability. The incorporation of the reflexive command filter as a means to protect the system from higher-level software errors that would otherwise result in collisions with obstacles adds another dimension to the research area of fault tolerance and system reliability.

The reflexive command filter accepts commands from higher, more intelligent layers, monitors the state of the system, inspects the local configuration space, and anticipates potential damage to the system. The reflexive command filter continuously and rapidly projects whether it is necessary to prevent a collision. On each reflex cycle, if it can be proven that a protective reaction

may be safely deferred, then the higher-level motion command is accepted and executed verbatim. If, however, the reflexive command filter cannot prove that protective action may be postponed, then it substitutes a protective action in place of the higher-level command. The protective action generated by the reflexive command filter is as close to the commanded action as possible, subject to the constraint that no damage may occur.

The reflexive command filter has been shown to be efficient, permitting full-speed robot operation while ensuring collision prevention. The reflexive command filter has been tested with higher layers of both teleoperation and on-line motion planning. In the case of teleoperation, the reflexive command filter permitted the robot to respond to operator commands up to the full bandwidth of the actuators, while simultaneously protecting the robot from even deliberately malicious motion commands. In the case of on-line planning, the reflexive command filter actually improved the performance of the planner, as the planner did not need to concern itself with high resolution or with possible danger from dynamic tracking errors.

It should be noted that the reflexive command filter not only prevents obstacle collisions but, as is illustrated in Figure 5.7, also is capable of some navigation. However, the reflexive command filter is in general less competent than, for example, deliberately chosen potential functions in finding simple

paths around obstacles. This can be explained in terms of energy. If a joint-space goal point, $\boldsymbol{q}_{\text{goal}}$, is represented as an attractive potential function, then we can evaluate the energy associated with any pose of the robot, $\boldsymbol{q}$, as $E = \sum_{i=1}^{N} E_i$ where $E_i = \frac{1}{2} K_i (q_{\text{des}_i} - q_i)^2$, $K_i > 0$. In Section 5.3 it was mentioned that if an obstacle is found on a search front the expansion of the search front in the corresponding direction is halted. For our reflexive command filter, this means that joints are controlled in a manner which enforces $d(E_i)/dt \leq 0$ for *each* joint i. Alternatively, a typical potential-function controller would execute a path corresponding to the steepest descent of the *total* energy, E. At an obstacle boundary, moving into an obstacle is equivalent to a steep positive energy gradient. As a result, the steepest descent of E near an obstacle might well require an *increase* in one or more individual joint energies, E_i. Thus, potential-function controllers can often continue to make progress towards a goal ($dE/dt < 0$) in cases where the reflexive command filter alone would stall ($d(E_j)/dt > 0$ for some j). In Section 7.2 we describe how to to augment the planning behavior of the reflexive command filter, using "guiding potential functions."

5.6 Conclusions and Overview for Chapter 5

The reflexive command filter accepts commands from higher, more intelligent layers, monitors the state of the system, inspects the local configuration space,

and anticipates potential damage to the system. The reflexive command filter continuously and rapidly projects whether it is necessary to prevent a collision. On each reflex cycle, if it can be proven that a protective reaction may be safely deferred, then the higher-level motion command is accepted and executed verbatim. If, however, the reflexive command filter cannot prove that protective action may be postponed, then it substitutes a protective action in place of the higher-level command.

The reflexive command filter exhibits the following virtues: It does not suffer from unrealistic or overly restrictive assumptions on robot dynamics; its complexity does not increase with higher dimensions; it does not fail or slow down as environment complexity increases; it does guarantee collision avoidance (at least for static obstacles) and can be computed on line; it does not suffer from excessive influence of obstacle repulsion fields; its complexity increases only linearly with higher dimensions; it is ideally transparent, i.e., it does not introduce any significant distortion of higher-level controls in all but emergency cases; and it fits in control hierarchies.

The reflexive command filter consists of two sub-modules which operate asynchronously. One module is the *C-space inspector* and the other is the *active command filter*. The C-space inspector corresponds to the reflexive command filters virtual sensor and the active command filter is the active component of the reflexive command filter.

The C-space inspector produces an obstacle-free C-space prism, which the active component uses to determine whether the currently incoming command should be approved or replaced with a predefined braking policy. A crucial construct of our reflexive collision avoidance scheme is the concept of the *braking prism.* The braking prism contains the robot's trajectory resulting from enforcement of a braking policy. If it can be proven that the braking prism will stay within the free space prism during the next execution cycle if the current command is approved, then the current command will be approved. If this is not the case the current command will be replaced by a braking policy.

The primary objective of the reflexive command filter is to guarantee system survivability. The reflexive command filter is particularly useful when new applications programs are being tested, or when a robot is teleoperated. However, it has also been proven to be an ideal local operator in subgoal based path-planners.

Chapter 6

The Reactive Trapezoidal Trajectory Generator

The reactive trapezoidal trajectory generator accepts position commands and generates continuous trajectory commands. These commands are issued as velocity commands when our servo-controller is a pure velocity controller, or as position, velocity, and acceleration commands in the more general case. The trajectory generator operates by planning a trapezoidal angular velocity profile for each joint. Slope and height constraints on the trapezoidal velocity profile restrict resulting trajectories to conform to the machine's joint-by-joint acceleration and velocity limits. Thus, all trajectories generated can be executed feasibly by the underlying tracking controller. At the same time, though, the trajectories generated are nearly time-optimal. A crucial new feature of our trajectory generator is that it is "reactive." That is, it is not necessary to complete any one planned trajectory before responding to new position commands. New trajectory plans are continuously recomputed with initial conditions equal to the current robot state, and goal conditions equal to the

current position command. This feature is essential if the robot is working under reflex control.

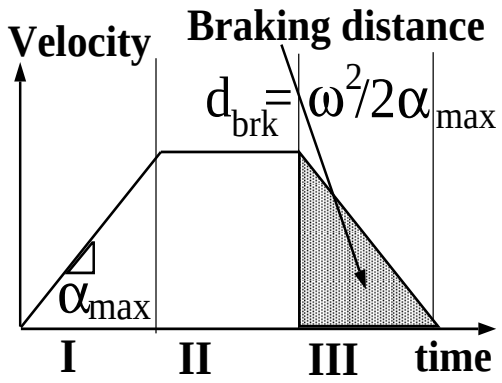


Figure 6.1:

The reactive trapezoidal velocity profiler first determines whether the robot should accelerate, move with a constant velocity or brake. Then it generates new trajectory commands based on the current state of the robot. In the case where the reactive trapezoidal trajectory generator generates commands to a PD-DD (position-velocity-acceleration) controller, it provides the following control law for each joint (The regions refer to Figure 6.1).

- Region III : If $\partial|d|/\partial t < 0$ and $|d| < d_{brake}$, where $d = \theta_{des} - \theta$ and d_{brake} is the braking distance, assert maximum braking. The maximum braking is asserted by generating the following commands to a PD-DD controller:

$$\begin{aligned} \alpha_{des} &= -\mathbf{sgn}(d)\alpha_{max} \qquad (6.1) \\ \omega_{des} &= \mathbf{sgn}(d)\sqrt{2|d|\alpha_{max}} \\ \theta_{des} &= \theta + \omega_{old}\Delta t - \end{aligned}$$

$$\mathbf{sgn}(d)\alpha_{max}\Delta t^2/2$$

where α_{des} is the desired acceleration, α_{max} the maximum acceleration, ω_{des} the desired velocity, θ_{des} the desired position, $\omega_{old} = \omega(t - \Delta t)$ the last velocity command, and Δt is the loop time of the servo program, or in other words the time that elapses between command updates.

- Region II : else if $\omega > (1 - \epsilon)\omega_{max}$ assert constant velocity, where ω_{max} is the maximum velocity and ϵ is a small constant that is adjusted for adequate noise immunity. Constant maximum velocity is asserted by generating the following commands to the PD-DD controller.

$$\begin{aligned} \alpha_{des} &= 0 \\ \omega_{des} &= \mathbf{sgn}(d)\omega_{max} \\ \theta_{des} &= \theta + \mathbf{sgn}(d)\omega_{max}\Delta t \end{aligned} \tag{6.2}$$

- Region I : else assert maximum acceleration. This will happen if we are moving in the opposite direction to where we want to be or the velocity is less than ω_{max} and $|d| \geq d_{brake}$. Maximum acceleration is asserted by generating the following commands to the PD-DD controller.

$$\begin{aligned} \alpha_{des} &= \mathbf{sgn}(d)\alpha_{max} \\ \omega_{des} &= \mathbf{sgn}(d)\alpha_{max}\Delta t + \omega_{old} \\ \theta_{des} &= \theta + \omega_{old}\Delta t + \end{aligned} \tag{6.3}$$

$$\mathbf{sgn}(d)\alpha_{max}\Delta t^2/2$$

In the case where the reactive trapezoidal trajectory generator generates commands to a velocity controller, it provides the following control law for each joint:

- Region IV : If $|d| < \sigma$, where $\sigma = 0.5^o$ in our experiments, generate a velocity command that results in the same commanded torque as a fictitious PD controller. The purpose of this is to ensure that the robot is brought close to the goal. We assume that the velocity controller generates torque commands according to $\tau_{cmd} = K_v(\omega_{des} - \omega)$, and that the PD controller used in our comparison is $\tau_{cmd} = K_p(\theta_{des} - \theta) + K_v(\omega_{des} - \omega)$. The PD-equivalent velocity command is asserted by generating the following velocity command :

$$\omega_{des} = K_p d/K_v \tag{6.4}$$

- Region III : else if $\partial|d|/\partial t < 0$ and $|d| < d_{brake}$, assert close to maximum braking. The maximum braking is asserted by generating the following velocity command :

$$\omega_{des} = \mathbf{sgn}(d)\max(\sqrt{2|d|\alpha_{max}}, \sigma K_p/K_v) \tag{6.5}$$

- Region I,II : else assert maximum acceleration or maximum velocity by

generating the following velocity command :

$$\omega_{des} = \mathbf{sgn}(d)\omega_{max} \tag{6.6}$$

The reactive trapezoidal trajectory profiler is smooth, non overshooting, nearly time optimal, stable, adjustable, updates quickly, and is amenable to quickly changing position commands, and is therefore ideal in the context of reflex control. For the same reasons the reactive trapezoidal trajectory generator allows construction of systems which are capable of exhibiting stimulus-responsive behaviors. The dominant class of machines outside the laboratory utilizes velocity control at the analog level, thus a reactive trapezoidal trajectory profiler generating commands to a velocity controller is applicable to a broad range of machines. In Figure 6.1 we illustrate the general behavior of the reactive trapezoid.

Chapter 7

On-line Path Planning and Obstacle Avoidance Using Reflex Control

This section will discuss the use of reflexive obstacle avoidance in conjunction with higher-level obstacle avoidance and path planning methods. Motion planning and obstacle avoidance are important aspects of robot autonomy. Without a motion planner for robot's, human operators have to specify every detail of a robots motion, which can be very difficult or even impossible in changing or dynamic environments. However, automatic obstacle avoidance and motion planning is a very complex task. In this chapter I will give a brief overview of different approaches to path planning 7.1, and then describe a few path planners utilizing reflex control 7.2,7.5,7.6. A more extensive overview of the different approaches in the field is given in [43] and [34].

7.1 Overview of Path Planning Methods

Motion planning methods in static environments are usually grouped into three types of approaches :

- The road map method or the skeleton approach. In this method the free C-space is retracted to a network of one-dimensional lines. Examples of this approach are :

  - The visibility graph. The visibility graph is a collection of lines in the free space that connects a feature of an object to that of another. Figure 7.1 illustrates the visibility graph of polygons, where vertices are used as features. This method is one of the earliest path planning methods [60] and it has been widely used to implement path planners for mobile robots.

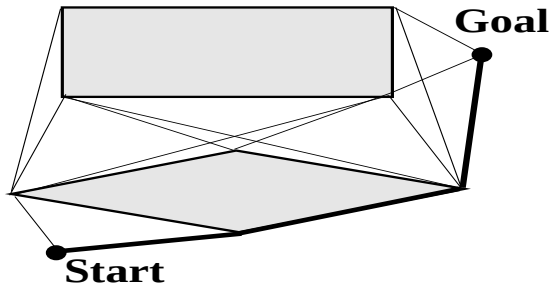


Figure 7.1: Visibility graph. Solution path is shown in bold lines.

  - The Voronoi diagram. The Voronoi diagram is the set of points that are equidistant from two or more object features. Figure 7.2 illustrates the Voronoi diagram graph of polygons, where the polygon edges are used as features. This method was first introduced in [61].

  - The silhouette method. In the silhouette method higher-dimensional objects are projected to lower dimensional objects and then traces

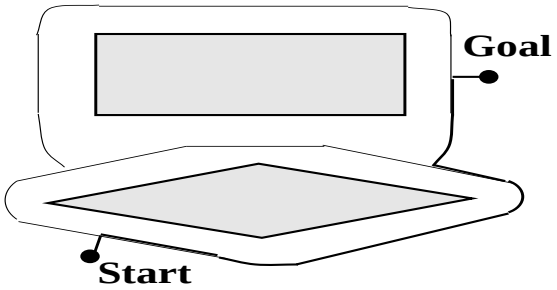


Figure 7.2: The Voronoi diagram. Solution path is shown in bold lines.

out the boundary curves of the projection. The silhouette curves are recursively projected to a lower-dimensional space, until they become one-dimensional lines. The curves are then connected at places where new silhouette curves appear or disappear using linking curves. This method is useful in high dimensional C-space and has been used in 4D C-space. In Figure 7.3 2D and 3D examples of silhouette curves are shown. The silhouette method was developed in [22, 23].

– The subgoal network method. A list of reachable configurations from start to goal is maintained. The reachability of one configuration from another is determined by a local path planner called *local operator*. Moving along a straight line is an example of a local operator. In this method the goal and the start position is connected to a subgoal with the help of local operator. Next a sequence of subgoals is found which connects start and goal. The local operator is then used to navigate between the subgoals. This method was introduced in [28]

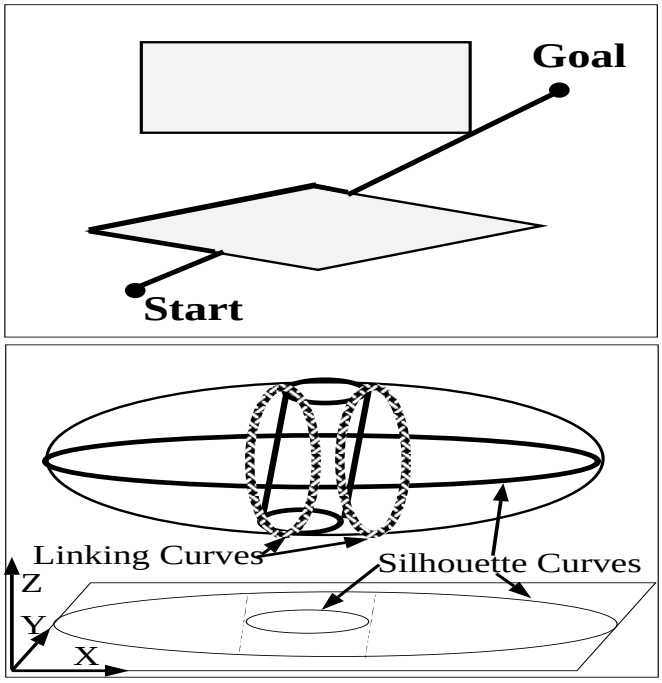


Figure 7.3: Silhouette method in 2D and 3D. The silhouette curves in the 2D example are the boundary curves of the polygons. The linking curves from are straight ines from start and goal. The ellipsoid in the 3D example has a cylindrical hole. It is projected on the X-Y plane, with the resulting silhouette curves marked with a black solid curve. The linking curves shown connects the silhouette curves for the cylinder with the silhouette curve for the ellipsoid.

and completed in [25]. Other local operators could be potential functions or a reflex controller. This method can be used for robots with a large number of degrees of freedom. An illustration of a subgoal network with straight line moves as a local operator is shown in 7.4.

– The freeway method. In the freeway method a generalized cylinder of free-space is found in the robots work space. This robot moves along the spine of this generalized cylinder. It is predetermined whether it is possible to rotate at the different locations along the spine, and

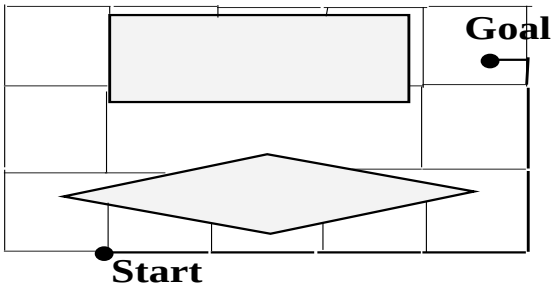


Figure 7.4: Subgoal network using straight line segments as a local operator.

this information is used in the path-planning step. The method was introduced in [14] and is used for mobile robots.

- Cell decomposition approaches. In this method the C-space is decomposed into a set of simple cells, and the adjacency relationships among the cells are computed. The cells containing the start and the goal are identified and then the adjacency relationships connect the two cells.

  - Object-dependent decomposition. The boundaries of the obstacles are used to generate the cell boundaries. The number of cells is small, but the complexity of the decomposition and adjacency computations are high. Object-dependent cell decomposition is illustrated in Figure 7.5. Different approaches to object-dependent decomposition are described in [66, 67, 3].

  - Grid-decomposition. C-space is decomposed into a large set of free or taken cells. This is illustrated in Figure 7.6.

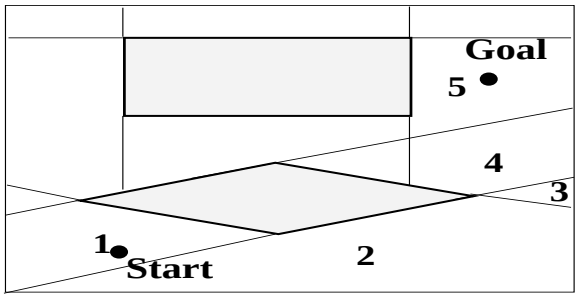


Figure 7.5: Object dependent cell decomposition.

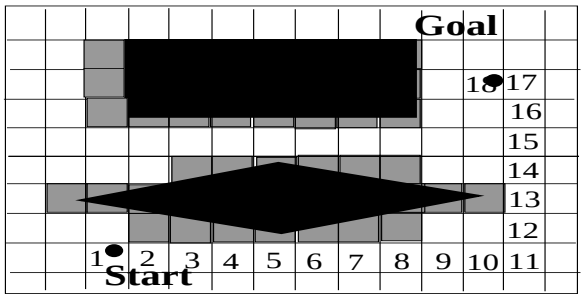


Figure 7.6: Object independent cell decomposition, using a grid.

  - Octree or quadtree decomposition. C-space is decomposed into cells which are either free, taken or mixed. The mixed cells are further decomposed into smaller cells.

- The potential field approach. In this approach a scalar function called the potential is constructed. The potential function has a minimum at the goal configuration and a high value around obstacles. The potential field approach is used for both obstacle avoidance and global path planning. The idea of using potential functions for obstacle avoidance was used in [37] and in [32] for force control. It was also developed independently in

[52] and [63]. When a potential function is used for local obstacle avoidance the virtual force resulting from the potential function is computed and applied to the motors.

An example of how to use potential functions for global path planning is the gradient approach. The gradient of the potential field is computed and a segment is generated in the direction of the gradient. From the new position a new segment is computed, etc. Another approach is the grid approach. A grid is constructed with a specific potential function value. A path is then generated searching the surrounding cells for lower potential function values. Potential functions can either be precomputed and formed into a grid of function values or computed on-line from obstacle features. In the latter case you get a continuous potential function. However, in the latter case it will also be more difficult to retrieve the potential function value in a time efficient fashion.

A common problem when using potential functions for global path planning is that they often result in local minima. Another problem is that potential functions usually do not consider local dynamic behavior. A variety of potential functions have been proposed in the literature which are aimed at either improving local dynamic behavior, or reducing the number of local minima and/or the size of their attractive wells. One example of a potential function that attempts to improve local dynamic

behavior is the generalized potential field [41] which is a function of both the robot's configuration and velocity.

A potential function with a minimum located at the goal and whose attractive domain includes the entire free space is called a global navigation function. In [38] Koditschek showed that in the presence of C-obstacles a global navigation function does not exist in general. If there are N disjoint C-obstacles the Potential function must possess at least N saddle points. However, it is possible to construct potential functions with a minimum at the goal position whose domain of attraction includes all free space connected to the goal position, except a set of points which are saddles of the potential function. Such a potential function is called an "an almost global navigation function" or simply a *navigation function.* In [65] Rimon and Koditchek developed a method for generating navigation functions.

- Numerical Navigation Functions : Its is very difficult to construct an analytical navigation function over a free space of arbitrary geometry. However, the computation of a numerical navigation function over a representation of the configuration space in the form of a grid turns out to be a much easier problem. In [4] Barraquand and Latombe proposed a method for computing numerical navigation functions. A wavefront is

expanded starting at the goal. This wavefront cannot traverse obstacles, instead it is diffracted around obstacles in a similar fashion to real waves. Finally this wave will reach the start position if a path exists between start and goal. To generate a path the wave is followed backwards towards "older" and "older" wavefronts until the goal is reached. This is illustrated in Figure 7.7. The navigation function is free from local minima and therefore guarantees a solution if one exists. Unlike other potential function methods the numerical navigation function is computed globally. This means that it will find a path to the goal if one exists, but it also means that the computation is more time consuming than it is for local potential function methods.

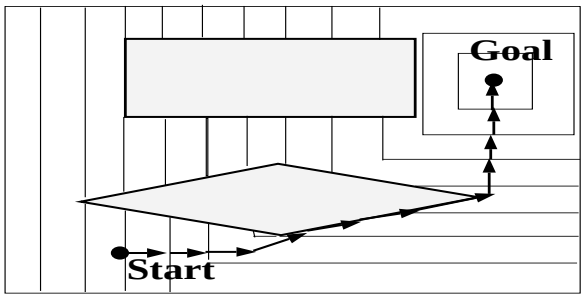


Figure 7.7: Illustration of navigation function.

In this section I will describe how reflexes can enhance the performance of path planning methods and also make it easier to construct efficient path planning methods. One way reflexes enhance higher-level path planners is that the reflex controller guarantees obstacle avoidance taking into account the dynamics of the system. Very few path planning methods take into account the

dynamics of the robot system and those which do are either computationally extremely complex or they invoke extremely conservative approximations. Reflex control is also ideal as a local operator in the subgoal-based path planning approach. Reflex control further allows for large deviations from pre-defined paths without this resulting in collisions. This can be used to construct nearly time optimal path planning algorithms which do not attempt to generate exact paths. It should be noted that the reflex controller also can be used as a safety device which protects the robot from collisions in case the higher-level path planning algorithms generates dangerous commands. I will in the following give examples of efficient obstacle avoidance and path planning algorithms which rely on the reflex controller for their performance.

7.2 Obstacle Avoidance Using Reflex Control and Guiding Potential Functions

Guiding potential functions are local potential functions which are applied in conjunction with the reflexive command filter, and computed on-line. The purpose of the guiding potential is to enhance the navigation capability of a system utilizing the reflexive command filter, without adding higher level planning capabilities to the system. Our guiding potential function alleviates

some of the restrictive behavior of our reflex controller. The current improvement can be explained best in terms of energy. If a joint-space goal point, $\boldsymbol{q}_{goal}$, is represented as an attractive potential function, then we can evaluate the energy associated with any pose of the robot, $\boldsymbol{q}$, as $E = \sum_{i=1}^{N} E_i$ where $E_i = \frac{1}{2} K_i (q_{des,i} - q_i)^2$. Under our reflex control, joints are controlled in a manner which enforces $d(E_i)/dt \leq 0$ for *each* joint i. Alternatively, a typical potential-function controller would execute a path corresponding to the steepest descent of the *total* energy, E. At an obstacle boundary, moving into an obstacle is equivalent to a steep positive energy gradient. As a result, the steepest descent of E near an obstacle might well require an *increase* in one or more individual joint energies, E_i. Thus, potential-function controllers can often continue to make progress towards a goal ($dE/dt < 0$) in cases where reflex control alone would stall ($d(E_j)/dt > 0$ for some j).

The effects of our reflex control and potential-function control are contrasted in Fig 7.8. In this figure, there is a smooth obstruction between a robot position and the goal. Under the influence of a potential function, the robot would be attracted to the goal along a path of continuously decreasing total energy, and it would succeed in circumventing the obstacle. Our joint-by-joint reflexes, on the other hand, would cause the robot to stall at the point indicated. Further motion from here would result in an increase in E_2, the energy associated with link 2, and is thus forbidden by the reflexes.

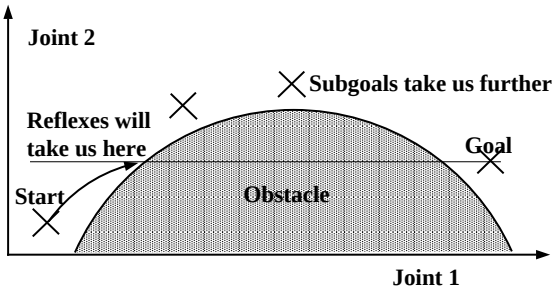


Figure 7.8: The reflexive command filter will not overshoot any individual joint goal en route to the final goal

Although our joint-by-joint implementation of the reflexive command filter is less competent than potential functions, its implementation is highly efficient and dependable. To enhance competence, we added a guiding potential function as a higher layer. This higher layer generates subgoals equivalent to the path which would have been taken by a potential-function controller alone. These subgoals are submitted to the reflexes, thus guiding the robot through a path which is incrementally achievable under reflex control. Thus the guiding potential function layer imitates the action of a potential function but relies on the reflexes to protect the robot from colliding with obstacles. In general we can construct a guiding potential function for any purpose– to get around obstacles, to start avoiding them at an early stage, etc. The important thing to note is that a guiding potential function relies on the reflexes for guaranteed obstacle avoidance, and details of robot dynamics may be ignored by the higher-level guiding potential function. Since the reflexive command filter absolutely prohibit collisions, any guiding potential function may be used,

regardless of the guiding function's inability to guarantee collision protection (e.g., we may use [38],[36], or [2]).

In our implementation, we have investigated three guiding potential functions:

- A *configuration-space guiding potential function* that computes energies as a function of distance in joint-space to a specified joint-space goal, $\boldsymbol{q}_{des,i}$, assuming a virtual joint stiffness matrix K_q:

  $E_{CS} = \sum_{i=1}^{N} K_{q,ii}(q_{des,i} - q_i)^2$ In this case the guiding potential function results in the same path as the joint by joint reflex controller when no obstacles are present. When at the boundary of an obstacle, though, the guiding potential function leads the robot around obstructions when such motion would decrease total energy.

- A *workspace guiding potential function* that computes energies as a function of distance to a specified hand pose, $\boldsymbol{x}_{des}$ (with a large workspace stiffness K_x), plus a weak attraction to a preferred joint pose, $\boldsymbol{q}_{des}$ (with weak joint-space stiffness K_q):

$$E_{WS} = \sum_{i=1}^{M} K_{x,ii}(x_{des,i} - x_i)^2 + \sum_{i=1}^{N} K_{q,ii}(q_{des,i} - q_i)^2$$

  It is often more interesting to position the tool at a certain location in work space, rather than achieving a certain robot configuration. This

guiding potential function is active everywhere in space and generates subgoals that are closer to the workspace goal. Independent of where in space the robot is located this layer tries to find voxels in the vicinity of the current position that do not belong to obstacles and are closer to the goal in work space than the current position. This layer will thus generate subgoals to the reflex layer that will take the robot around obstacles in a stable manner. This guiding potential function guides the robot in the entire space, and is therefore not minimally influential.

- An *augmented task-space potential function* that computes energies as a function of distance to a specified task-space goal, $\boldsymbol{y}_{des}$ consisting of the hand pose and an elbow angle :

$$E_{TS} = \sum_{i=1}^{N} K_{y,ii}(y_{des,i} - y_i)^2 \tag{7.1}$$

  By using augmented task-space coordinates, the joint space term is eliminated.

where $\boldsymbol{x}_{des}$ is an M-dimensional desired workspace goal, and $\boldsymbol{q}_{des}$ is an N-dimensional desired joint-space goal.

In the case of the configuration-space potential function, the guiding potential function results in the same path as the joint-by-joint reflex controller when no obstacles are present. When at the boundary of an obstacle, though,

the guiding potential function leads the robot around obstructions when such motion would decrease total energy.

In the case of the workspace potential function, the arm path is defined instead by attraction to a hand goal. A weak joint-space potential function is added to the hand energy to prevent erratic null-space motions.

We note that it is feasible (and often useful) to bypass the guiding potential function, or to switch in an alternative guiding potential function spontaneously, and we may do so without danger of an ensuing collision. However, if one chooses to switch control modes, the result can be robot trajectories which cycle in an infinite loop or which chatter. From an energy perspective, one can evaluate the stability of prospective switching laws. In each alternative potential-function mode, we define energy gradients with respect to the selected mode's measure of energy. We may switch among multiple modes and still guarantee no sustained cycling or chattering provided: 1) all energy measures have a global minimum at the same goal state (i.e., all modes agree on the ultimate objective); and 2) the energy measure of a mode to be switched in is lower than the energy measure of that mode when it was last switched out. (We may initialized all mode energies to infinity). Under these conditions, it is impossible to revisit any state under the same control mode, thus excluding the possibility of sustained cycling or chatter.

In our experiments, we have found mode-switching to be useful in achieving

faster robot motions when the robot is sufficiently far from obstacles. Since the reflex control evaluations permit full-speed robot motion in free-space, which is generally faster than competing potential function evaluations, we choose to bypass the guiding potential functions when they are not needed. We may do so without risking collisions, and, provided we abide by the preceding mode-switching energy constraints, without stability problems.

7.3 Implementations of Potential Function Layers in Discretized Configuration Space

Although we have stated a conceptually satisfactory description of guiding potential functions, we encountered practical problems in implementation. A significant problem for energy-gradient-based methods acting in a discretized space is that discretization causes false local energy minima and maxima, as illustrated in Fig 7.9. In this figure, a goal is located at the center of a circular obstacle. In a continuous space, we would measure the same energy everywhere on the obstacle boundary. In discretized space, we forbid entry into any cell which contains any part of an obstacle, and free-space cells are assigned a single energy corresponding to the coordinates of the center of the cell. Fig 7.9 shows several of the false energy minima and maxima which result from discretization.

To mitigate the discretization problem, our potential-function module is implemented as follows. For each robot state, we continuously compute the goal attraction energy for all free-space voxels in an N-space hypercube centered on the current robot state. (In our implementation, we normally inspect a hypercube of 7^4 voxels in 4-D C-space). The voxel with the lowest attractive energy within the search volume is selected as a subgoal. Note that forbidden voxels are excluded from the evaluation; thus the potential-function search becomes *more* efficient as the number of obstacles increases. Since our search selects the lowest-energy voxel within a defined neighborhood as a subgoal, it is possible to "tunnel" through a limited number of higher-energy voxels en route to a lower energy voxel. By considering energies of voxels which are not immediately adjacent to the current robot state, our guiding potential function is based on some smoothed neighborhood energy gradient, which helps reduce the ill effects of C-space discretization.

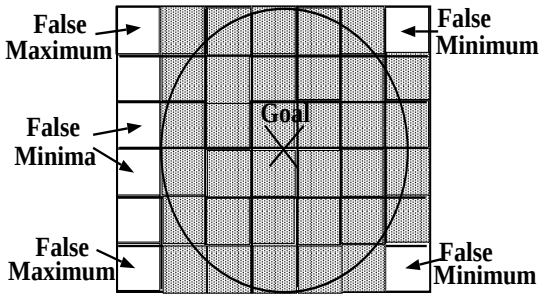


Figure 7.9: False Energy Minima and Maxima Due to C-space Discretization

7.4 Experiments on a Kinematically Redundant Industrial Robot

The system described here was implemented on the first four joints of the Robotics Research Corp K-2107HR robot arm. Our task-space commands consisted of the desired Cartesian coordinates of the wrist, which left an "elbow orbit" redundant degree of freedom available for obstacle avoidance. Both the joint-space and the workspace guiding potential functions described above were tested in coordination with reflexes, and the robot showed an ability to avoid collisions and circumvent obstacles quickly. Finally, and most importantly, a guiding potential function permits task specification in a lower dimension than the joint space. The robot's kinematic redundancy is then automatically utilized by the reflexes for on-line obstacle avoidance.

Figure 7.10 illustrates a recorded example of the reflex-controlled robot moving in a cluttered environment using subgoals generated by a work-space based guiding potential function. With respect to Figure 7.10, the top picture illustrates the robot in pose I. From this initial pose, the robot is commanded to reach each of three new poses, poses II, III, and IV, as shown. "Snapshots" of the robot's resulting path in C-space are illustrated in the scenes to the right. These scenes display 2-D C-space slice projections, where each slice projection shows forbidden poses as a function of two joint angles with the remaining two

joint angles fixed. In scenes of θ_3 vs θ_4, the "stalactite" extending from the top of the frame corresponds to the coat rack, and the island blob corresponds to the box obstacle. The pose of the robot in each instance is at the intersection of the crosshairs. The barcharts to the right display the robot's actual joint angles, in degrees, corresponding to the displays of C-space.

In the first move, the robot successfully moves from Pose I to Pose II through intermediate configurations of II.A, II.B and II.C, where II.C corresponds to Pose II. The robot reaches the Pose II goal wrist state by utilizing its null-space mobility, and orbiting its elbow to avoid the coat rack.

In the next sequence, the robot again starts from pose I, and it is commanded to place its wrist at a point on the opposite side while negotiating obstacles of the coat rack, a box, the floor, and joint limits. Scenes III.A, III.B and III.C show intermediate poses of its path, where III.C corresponds to the illustrated goal state Pose III.

Finally, the robot is instructed to move from Pose I to Pose IV, where Pose IV corresponds to a wrist goal inside a box. C-space images IV.A, IV.B and IV.C illustrate the resulting path; IV.C corresponds to the goal Pose IV. It is interesting to observe that this desired robot pose corresponds to a small, confined "hole" of freespace surrounded by obstacles in most C-space views. Although a seemingly complex path solution is achieved, it was obtained from our very low-level energy-based controller.

7.5 Obstacle Avoidance Using Reflex Control and Quickly Computable Continuous Potential Functions

Potential functions are usually either precomputed and stored as grids of data or computed on line from obstacle features and the goal location. In the first case it is very easy to retrieve the value of a potential function on line. The potential function will in this case be discretized. In the second case the potential function will be continuous. However, for complex or multiple obstacles it will be a time consuming operation to retrieve the value of the potential function. As mentioned above the potential function can be used for both on-line obstacle avoidance and global path planning, where the global path planning is done either by searching a potential function grid or using the gradient method.

In this section I will describe a potential function which I used for global path planning in an early version of my sonar-based world mapping system described in 9. On-line obstacle avoidance was performed by the reflexive command filter. This potential function is partly precomputed and partly computed on line, it is continuous and very easy to compute on-line. In the precomputation step discrete waves are generated around the obstacles, and the "wave voxels" in the wave fronts are numbered in accordance with their

distance to an obstacle. This is illustrated in Figure 7.11. A continuous potential is then found on-line from the wave voxel number at the current location and the wave voxel number of the neighboring wave voxels, and the current location within this wave voxel. The computed value will be a non-Euclidean measurement of the distance to the closest obstacle. The distance measurement used is $\min(\theta_1, \theta_2, \theta_3, \theta_4)$ or $\min(x, y, z)$ depending on whether the potential function is applied in C-space or the work space. The procedure for computing the potential function value on-line can be summarized as described below. Please refer to Figure 7.12 for a supplementary illustration.

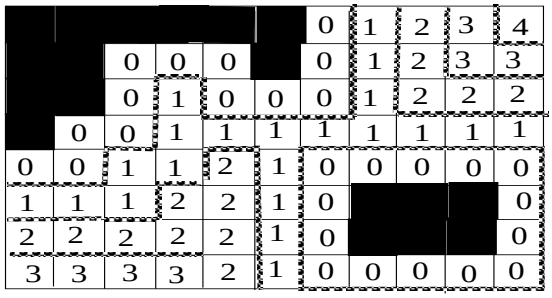


Figure 7.11: Illustration of wave voxel values around two obstacles.

1. Read the wave voxel value of the current position.

2. Find the distance $\min(\theta_1, \theta_2, \theta_3, \theta_4)$ or $\min(x, y, z)$ to the neighboring wave voxel with the lowest wave voxel number. If several neighboring wave voxels have the same value pick the overall shortest distance. Add the fraction, of this distance and the voxel size, to the current wave voxel value. Now you have a non-discrete measurement of the distance to the

closest obstacle, derived from discrete local information.

3. Use this distance measurement to form a continuous potential of desired type.

It should be noted that this potential does not contain any local minima.

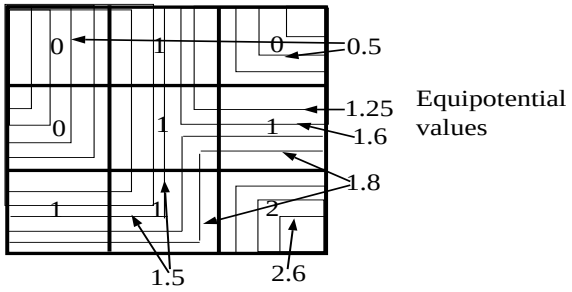


Figure 7.12: The distance measurements are illustrated in this figure as wave-like equipotential lines. The potential function is easily derived from current wave voxel value and the neighboring wave voxel values. The resulting function is continuous and does not contain local minima.

The paths generated by the potential function were filtered through the reflexive command filter, and in the instances the paths were flawed (development stage) the paths were rejected, and in the instances the robot due to dynamics was unable to follow the paths generated by the potential function, the reflexive command filter protected the robot. This potential function was also used for non-planned on-line attraction and retraction from dangerous and intersecting places. In these instances the reflexive command filter protected the robot from the dangerous commands that occasionally were generated.

7.6 On-line Path Planning Using the Reflexive Command Filter

The 4-D on-line motion planning algorithm described here is an augmentation of the 3-D motion planning algorithm developed for monodextrous industrial robots described in [39, 40]. The efficiency of the path planning algorithm is due to the fact that the reflexive command filter is able to bring the robot between so-called critical points in configuration space in a nearly time optimal manner, while guaranteeing fail-safe collision avoidance.

The path planning algorithm uses 2-D slices of the discretized 4-D C-space. We chose to slice the 4-D C-space into θ_3–θ_4 slices along the θ_1 and θ_2 axes. A very important building block in our 4-D path planner is our reflex-based 2-D planner. The 2-D planner is able to bring the robot to a specified goal within a given 2-D C-space slice by generating a small set of subgoals achievable using the reflexive command filter. First the 2-D planner identifies the critical points within the 2-D C-space slice. These critical points are peaks of C-space obstacles as illustrated in Figure 7.13. These peaks are extremal points on the border curves of the C-space obstacles with respect to one of the joint coordinates. For reasons explained in Section 5.5, the reflexive command filter is unable to bring the robot around such peaks. If such peaks exists in the current 2-D C-space slice, and they represent impediments to the reflexive

command filter, then the 2-D planner uses these peaks to generate a set of subgoals that are given to the reflexive command filter. An example of how this can be done is described in [39, 40].

The first task of the 4-D path planning algorithm is to identify connected regions of free-space within each 2-D slice. These regions of free-space are called *F-regions* and are illustrated in Figure 7.14. The second task of the planner is, for every F-region, to identify which F-regions in the neighboring 2-D slices the F-region is connected to, and generate a table describing these connections. This table is called the inter-region graph. Two F-regions are connected if they belong to two adjacent 2-D slices of C-space, and have overlapping areas of free-space.

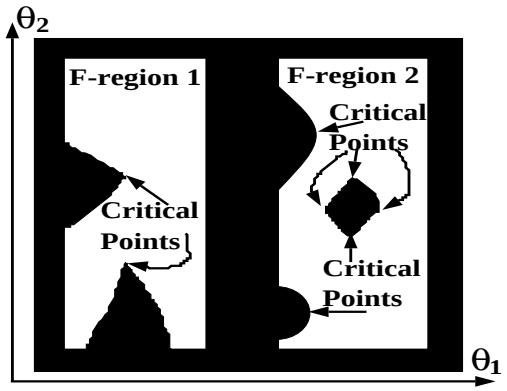
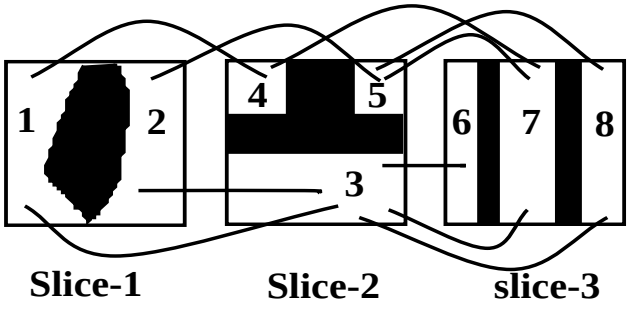


Figure 7.13: Reversal sets and F-regions.

Such connections are illustrated in Figure 7.14. If the current position of the robot is located in an F-region F_a, and F_a is connected to an F-region F_b, the robot could be moved to F_b by first moving the robot to an area where



F-region	connected to
1	3,4
2	3,5
3	6,7,8,1,2
4	1,6
5	7,8
6	4,3
7	3,5
8	3,5

Figure 7.14: Inter-region graph for 8 F-regions located in 3 adjacent C-space slices

F_a and F_b overlap, and at that point cross over from F_a to F_b. The third task of the planner is to identify the F-region where the goal is located, and the F-region where the current position is located, called F_{goal} and F_{start} respectively. Using the inter-region graph the planner then finds a path through the connected F-regions that connects F_{goal} to F_{start}, using a graph search method. The 4-D planner uses the 2-D planner to navigate within the F-regions. On execution the robot is moved from its start position within F_{start} to a point which belongs to an area overlapping with the next desired F-region. When this point is reached the robot crosses over to the next F-region, and repeats this operation until F_{goal} is reached. From there, the 2-D planner guides it

to the goal. Our 4-D motion-planning algorithm finds a solution whenever a solution exist. It is on-line in the sense that when new goals are requested, new paths are generated while the robot is moving. If new obstacles are brought into the robot's environment, a new C-space map must be computed and the corresponding F-regions identified. However, with our configuration space generator (described in Section 4) this can be done fairly quickly. Another important virtue of our path-planner is that it generates subgoals to the reflexive command filter instead of paths to a servo-controller, thus making it possible to utilize the full speed and acceleration capabilities of the robot, while maintaining obstacle avoidance.

7.7 Conclusions and Overview for Chapter 7

This chapter discussed the use of reflexive obstacle avoidance in conjunction with higher-level obstacle avoidance and path planning methods. First an overview of existing path planning methods was given. The path planning methods discussed were the visibility graph, the Voronoi diagram, the silhouette method, the subgoal network method, the freeway method, object-dependent cell decomposition, grid-decomposition, octree and quadtree decomposition, the potential field approach, and numerical navigation functions.

Especially the subgoal network method has received considerable ttention lately due to its efficiency [28, 25, 34]. The local operator in subgoal networks

usually consists of straight lines. However, introducing local potential functions as local operators has been discussed. This chapter suggest an alternative local operator, the reflexive command filter. If the reflexive command filter is used, the critical points are very easily defined, the critical points are few, and the robot does not necessarily have to follow a predefined path. Due to the high speed of the reflexive command filter and the fact that it guarantees obstacle avoidance, but still is transparent in all but emergency situations, this approach allows the robot to utilize its full acceleration and speed capabilities without invoking specific knowledge of the robot dynamics at the planning level.

The reflexive command filter in conjunction with guiding potential functions and quickly computable continuous potential functions was also discussed. Guiding potential functions are potential functions which are computed online locally, and which purpose is to enhance the navigation capability of the robot when using the reflexive command filter.

Chapter 8

Reflexive Avoidance of Moving Obstacles

To deal with obstacles approaching the robot we developed a *flee reflex*. The flee reflex generates commands to underlying levels which steer the robot away from an approaching object. Unlike the reflexive command filter which continuously approves or disapproves higher-level commands, the flee reflex remains inactive until an approaching object is detected. So far we have developed two types of flee reflexes:

- The *wall-emulating flee reflex* in which an approaching obstacle is modeled as an approaching wall. This flee reflex will be described in Section 8.1.
- The *block-emulating flee reflex* in which non-static obstacles are modeled as occupied blocks of work space. This flee reflex will be described in Section 8.2.

8.1 The Wall Emulating Flee Reflex

We tested the wall-emulating flee reflex in the context of two moving robots sharing a common workspace. In our implementation one of the robots, a

Hitachi PW-10, had a fictitious protective wall located at its tool position, while the second robot, a Robotics Research K-2107HR, avoided this wall. In our setup the wall moves in front of the PW-10 along the y-axis. The reflexive command filter protected the RRC from colliding with the fictitious wall, and the wall-emulating flee reflex repelled the RRC away from the fictitious wall when the wall touched and advanced towards the RRC. The experimental setup is illustrated in Figure 8.1. An overview of the system is given in Figure 8.2.

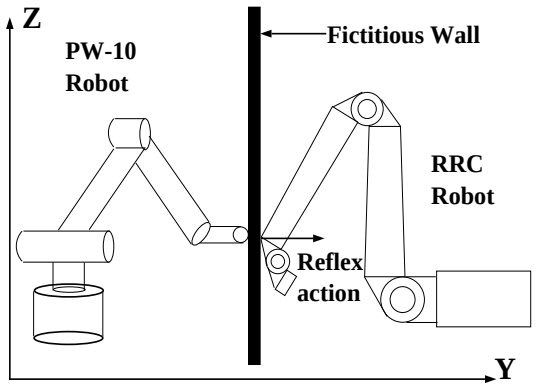


Figure 8.1: The fictitious PW-10 wall pushes the RRC-robot away.

In this implementation the wall-emulating flee reflex inspects the C-space and generates a fleeing motion if the C-space obstacle corresponding to the wall advances towards the RRC robot's current location. We precomputed the C-space obstacles corresponding to the fictitious wall located at different positions. These C-space obstacles are activated when the tool position of the PW-10 robot reaches the corresponding wall positions. This way the C-space

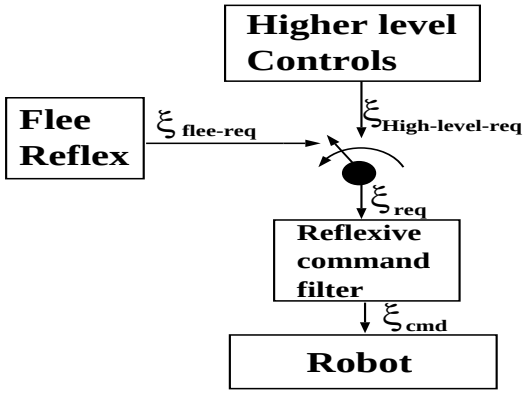


Figure 8.2: The flee Reflex in the system hierarchy. When an obstacle is detected, higher-level commands are replaced by flee reflex commands.

corresponding to the wall is updated at a rate of 700Hz.

As soon as the tool of the RRC intersects the fictitious wall, the wall-emulating flee reflex generates a *starle reflex*. The starle reflex quickly moves the robot away from the wall, but not necessarily in the direction of the y-axis. Next, the *retraction reflex* is generated. The retraction reflex moves the RRC away from the wall in the direction of the y-axis. The motivation behind using the starle reflex is that it can be calculated in 2-3 ms, and can, for that reason, be used to generate a protective action before the computations needed for the retraction reflex are completed. The computations for the retraction reflex take slightly less than 100ms. The starle reflex will remain active until the computations for the retraction reflex are done. The retraction reflex will remain active until the robot has moved away from the fictitious wall a distance corresponding to two voxels in C-space. As long as the fictitious wall is not approaching the RRC the reflexive command filter guarantees that the tool of

the RRC, and the fictitious wall never intersect.

The starle reflex is computed from the gradient of a *flee potential* with respect to the joint coordinates. The flee potential $\mathcal{F}(x, y, z)$ corresponds to a virtual force which is applied to the end effector of the robot. This virtual force pushes the end effector in the direction of the desired flee direction. In our case the RRC retracts in the direction of the y-axis, and for that reason we chose the flee potential to be $\mathcal{F}(x, y, z) = -k_{\mathcal{F}} y$. The constant $k_{\mathcal{F}}$ is in this case equal to the size of the constant virtual force applied to the end effector. $k_{\mathcal{F}}$ is chosen to be large enough to cause the robot to withdraw in the flee direction with an acceleration close to the joint acceleration limits. We have $\nabla\mathcal{F}(\theta_1, \theta_2, \theta_3, \theta_4) = J^T \nabla\mathcal{F}(x, y, z) = J^T(0, -k_{\mathcal{F}}, 0)^T$ where J^T is the Jacobian transpose. The gradient of the flee potential with respect to the joint coordinates results in a hand motion which is not parallel to the desired flee direction. Rather it corresponds to the direction which would result in the largest possible change of $\mathcal{F}$ with a small change of the joint coordinates. The starle reflex generates the joint velocity command vector $\boldsymbol{\omega}$ $= -k_\omega \nabla\mathcal{F}(\theta_1, \theta_2, \theta_3, \theta_4)$ to the robot (where k_ω is a velocity scale factor).

The retraction reflex moves the robot in a straight line along the y-axis. This is accomplished generating the joint velocity command $\boldsymbol{\omega} = -J_A^{-1} k_v \nabla\mathcal{F}(x, y, z, \phi)$, where in this case the inverse Jacobian is calculated using *augmented* task-space coordinates,as defined in [68], and $-k_v \nabla\mathcal{F}(x, y, z, \phi) = (0, v_{\text{flee}}, 0, 0)^T$.

The task-space coordinates we used were the three tool-position coordinates (x, y, z) and the angle (ϕ) between a predefined vertical plane and the plane containing the robot's shoulder, elbow and wrist as described in [68].

The reflexes discussed above allowed the RRC to share workspace area with the moving PW-10 robot, by preventing collisions between them. Since the reflexive command filter was activated at the same time, the RRC concurrently avoided all static obstacles.

8.2 The Block Set Emulating Flee Reflex

We also developed what I call the *block-emulating flee reflex*. In the block-emulating flee reflex moving obstacles are considered to occupy a set of world space blocks. When the PW-10 moves into the work space of the RRC, the body of the PW-10 will intersect a number of blocks in the work space of the RRC. This is illustrated in Figure 8.3. The blocks are predefined, and are 40cm wide and long and 60cm high. It is possible for the PW-10 to occupy 24 different blocks in the work space of the RRC. The arrangement of these 24 blocks is shown in Figure 8.4. The C-space for each block is precomputed and stored in global memory. The C-space corresponding to a particular block is activated when any part of the PW-10 intersects the corresponding block. This way several "block C-space" might be activated at any particular point. It should be noted that the C-space corresponding to the blocks, are not blocks.

Instead, the C-space are complex "snake looking" 4D constructions which to a large extent overlap each other. The program loop which determines which blocks the PW-10 intersects, runs at a speed of 40-50Hz on the same cpu as the PW-10 servo-controller.

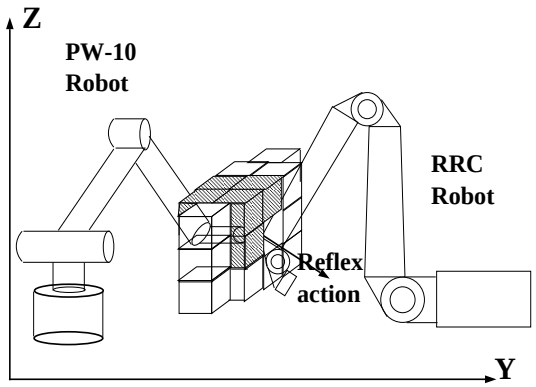


Figure 8.3: The PW-10 robot occupies different blocks depending on its position.

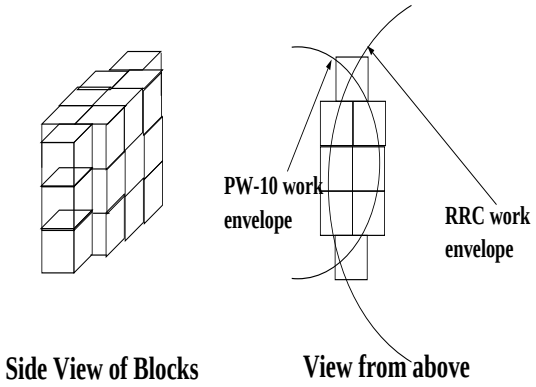


Figure 8.4: There are 24 blocks in the common workspace of the RRC and PW-10.

The desired flee direction is determined by the flee-potential field generated by the occupied blocks and the position of the RRC tool in this field. The flee-potential fields for each individual block are generated with respect to the

center point of each block. The individual flee-potentials are then superimposed on each other. There are two reasons for using the center point of the blocks as the center of the flee-potential.

1. The flee-potential only indicates the flee-direction. There is not a direct relationship between the gradient of the flee-potential and the torque generated to the servo amplifiers. An infinite potential at the surface of the cubes would not be of any help.

2. It is possible for the robot to find itself inside a block if, for example, the PW-10 very quickly invades the RRC robots workspace. In this situation we still would like the flee-potential to be defined.

The flee direction is computed using a potential function generated from the blocks. In this case the flee reflex also consists of a starle reflex and a retraction reflex. The starle reflex is computed from the flee potential using the following formula, $\boldsymbol{\omega} = -k_\omega \nabla \mathcal{F}(\theta_1, \theta_2, \theta_3, \theta_4) = -k_\omega J^T \nabla \mathcal{F}(x, y, z)$, and the retraction reflex is generated according to, $\boldsymbol{\omega} = -J_A^{-1} k_v \nabla \mathcal{F}(x, y, z, \phi)$, where the inverse Jacobian is calculated using *augmented* task-space coordinates.

The reach of the flee potential is determined by the *paranoia factor*. When the paranoia factor is high the RRC robot will stay away from the PW-10 at a safe distance. When the paranoia factor is low, the RRC is more brave in its attempts to get close to the PW-10. The reason behind introducing a paranoia

factor is that, the paranoia factor allows for adjustment of the system with respect to the speed and the degree of predictability of the moving obstacles.

The use of a flee-potential which reaches beyond the C-space obstacles corresponding to the world-space blocks, introduces a problem. When the flee reflex is turned off well outside the C-space obstacles, the robot will attempt to move closer to its goal again, which might mean that the robot will reenter the flee-potential, instead of colliding with a C-space obstacle. If the robot collides with a C-space obstacle the reflexive command filter will prevent further motion and keep the robot at a certain position. However, if the robot reenters the flee-potential before this happens, the flee-response will be reactivated and the result would be infinite chattering under static conditions.

To solve this problem we introduced the so called "hold reflex". The *hold reflex* is activated as soon as the flee reflex is turned off, and will keep the robot at the point where it was activated, unless a higher-level command is issued which would bring the robot away from the moving obstacles, or the flee potential is either retracting or expanding. The hold reflex can be classified as a command dependent reflex action. It should be noted that depending on the paranoia factor, and the coarseness of the conservative C-space approximation, it can be either the reflexive command filter, or the flee reflex which prevents the robot from colliding with a "temporarily static" or slow moving obstacle.

8.3 Conclusions and Overview for Chapter 8

To deal with obstacles approaching the robot, a flee reflex was developed. The flee reflex generates commands to underlying levels which steer the robot away from an approaching object. Unlike the reflexive command filter which continuously approves or disapproves higher-level commands, the flee reflex remains inactive until an approaching object is detected. Two types of flee reflexes were developed:

- The wall-emulating flee reflex in which an approaching obstacle is modeled as an approaching wall.
- The block-emulating flee reflex in which non-static obstacles are modeled as occupied blocks of work space.

Both flee reflexes were tested in the context of two moving robots sharing a common workspace.

The wall-emulating flee reflex consists of two "sub-reflexes": the retraction reflex and the startle reflex. The startle reflex is computed from the gradient of the flee potential with respect to the joint coordinates. This computation is very fast and takes a few milliseconds. The retraction reflex on the other hand moves the robot in a straight line motion along the y-axis, and takes roughly 100ms. The static command filter protects the robot from collisions under static conditions. The flee reflex is activated when the virtual wall

corresponding to the PW-10 robot is approaching.

The block-emulating flee reflex consists of the startle reflex, the retraction reflex, and the hold reflex. The startle reflex is, in this case, also computed from the gradient of the flee potential with respect to the joint coordinates. The retraction reflex on the other hand moves the robot in a direction derived from the gradient of the flee potential with respect to world coordinates. The block-emulating flee reflex has a paranoia factor associated with it. The paranoia factor determines the reach of the flee potential. When the paranoia factor is high the RRC robot will stay away from the PW-10 at a safe distance. When the paranoia factor is low, the RRC is more brave in its attempts to get close to the PW-10. Under static conditions it was either the static command filter or the hold reflex which controlled the robot. The reason the hold reflex was needed was to hold the robot in positions defined by a high paranoia factor which could be located far away from the actual C-space obstacle.

Both the wall-emulating flee reflex and the block-emulating flee reflex are on-line, very fast, simple in design, and therefore practical. They could be used as a means to prevent collisions in a multi-robot environment without having to rely on complex planning algorithms.

Chapter 9

Reflexes and Fixed Action Patterns in Sonar-based World Mapping

Reflexes and fixed action patterns can be used to build flexible, autonomous systems which are robust with respect to unpredictable and noisy environments. We have developed a sonar-based world mapping system in which reflexes and fixed action patterns are fundamental components. The experimental testbed consists of six Migatronics transducers mounted at the tool flange of a K-2107HR 7d.o.f. Robotics Research Corporation Manipulator. The transducer functions as both a transmitter and a receiver of ultrasound. The beam is cone-shaped with angle width of 30 degrees. The transducer was able to find obstacles placed between 2.5 inch to 4 feet away from the transducer.

Except for interpreting sonar data and generating maps, the system automatically generates an environment guided world exploration pattern. The robot is able to explore its environment without having explicit information about how to do this. The system is robust, flexible and allows for incremental

design. This is achieved by using behavioral modules like "reflexes" and "fixed action patterns".

The system initially assumes a predefined, possibly incomplete map of the robot's environment. The robot explores its environment and tries to find inconsistencies with this predefined map. The system also allows the robot to perform tasks while it continuously checks for inconsistencies. This way it is possible to use the sonar-based world mapping system either by itself, or in conjunction with other higher-level systems. A scheme similar to the methodology described in [5] is used to construct a map from the sonar readings. An overview of the system hierarchy with the sonar-based world mapping system as a module is shown in Figure 9.1.

An inconsistency is a reading by the ultrasonic sensor which cannot be explained as a reflection from an obstacle in the predefined map or from the robot itself, or as a specular reflection. The sonar sensor will not always detect all obstacles in the workspace. However, a detected inconsistency strongly indicates the existence of unmapped obstacles. The robot investigates unmapped obstacles by moving the sensor to different locations in the workspace while directing the transducer beam towards the location of the found inconsistency.

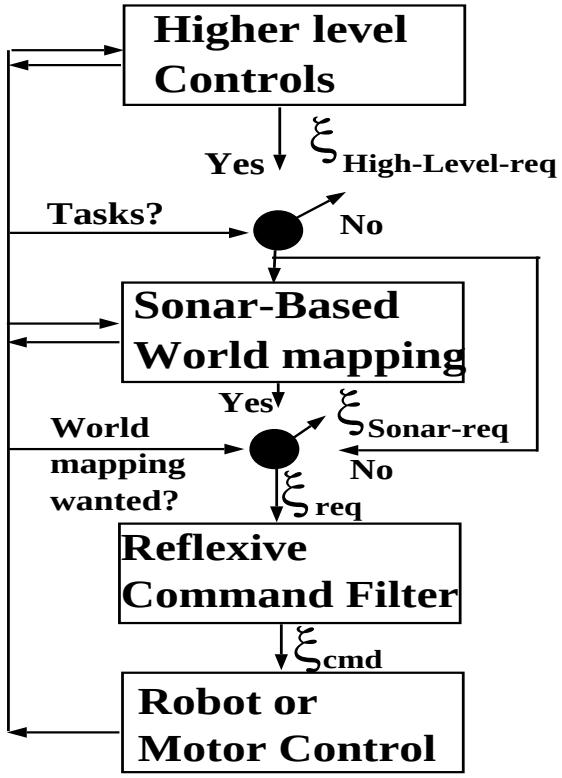


Figure 9.1: If the robot has a task to perform or a destination to go to, the robot attempts to complete these tasks, while the sonar-based world mapping system interferes to protect the robot from unknown obstacles. If the robot does not have a task to perform the sonar-based world mapping system completely controls the robot. In both cases the reflexive command filter protects the robot from collisions with known obstacles.

9.1 A System Overview

The system consists of a number of modules, one set of sensor interpretation modules, one set of world exploration modules, and one set of map generation modules. The sensor interpretation modules determines whether the sonar readings correspond to inconsistencies. The sensor interpretation modules are,

1. The *echo interpreter*, which uses the return time of the echo to generate a corresponding distance.

2. The *beam locator*, which uses forward kinematics to determine the location of the transducer, and to find the beam direction.

3. The *inconsistency determinator*, which based on the current world map and the echo distance, determines whether the reading corresponds to a reflection from a known obstacle, an invalid reading, or a reflection from an unknown obstacle, or an obstacle under investigation. If a reading corresponds to a beam which seems to have passed through a known obstacle, this probably corresponds to a specular reflection and the reading is declared invalid.

The world exploration modules are behavioral modules, such as fixed action patterns, or reflexes. These behavioral modules can be modeled as a virtual sensor active component pair. The information provided by the sensor interpretation modules, together with internal sensor information like resolver data etc., forms the different virtual sensors of the world exploration modules. The world exploration modules are,

1. *Look-Path*, controls the motion of the tool (joint-5-6). It attempts to direct the beam to areas in space where the robot is anticipated to be in the immediate future. It is active when the robot has a task complete.

2. *Look-Around*, controls the motion of the tool and the robot. It generates a robot motion in conjunction with a tool motion intended to eventually sweep all work space with the sonar beam. It is active when the robot has no task to complete.

3. *Beam-At*, controls the motion of the tool. It attempts to direct the beam to areas in space where inconsistencies are found. It is activated by new inconsistencies, and disactivated by "Investigate".

4. *Approach*, makes the robot carefully approach recently discovered inconsistencies. Approach is halted several times by the "Halt reflex" to make sure that, Beam-At is successful in examining the inconsistency from a distance. It is activated when Beam-At has been successful in one initial examination.

5. *Investigate* consists of a series of robot motions and tool orientations which places the sonar sensor at multiple positions and orientations in the vicinity of recently found inconsistencies. Investigate is guided by the sensor interpretation modules in determining the robots' motion. The sensor interpretation modules provides information about new inconsistencies, and a temporary map provides information about old inconsistencies. The distance, shape and size of the inconsistencies determines, and continuously updates the motion pattern generated by Investigate.

The system also contains a *Halt reflex* which temporarily stops the robot from proceeding. The Halt-reflex is activated when Approach is active, but also when the robot is trying to complete tasks in an unknown environment. The purpose of the reflex is to halt the robot for a while and give the system a chance to "beam-out" unfamiliar environments.

There are five map generation modules. Two which generate only temporary and local maps used for currently active modules like Approach, and Investigate, and three modules which generate three different types of long term maps. The five map generation modules are,

1. The *Initial-Navigation-Map-Generator* creates and continuously updates a "voxel" based map of recently found inconsistencies. This map is generated during the Beam-At and Approach phase. Voxels in the work-space are filled out as occupied in accordance with the sensor readings and the beam-width. This is an inexact conservative map which is used for navigation during the Approach and early Investigate phase. If the robot is reinvestigating an earlier found inconsistency, this map is only created and used if the new readings are inconsistent with the map for the earlier found inconsistency.

2. The *Temporary-Map-Generator* creates and continuously updates a "voxel" based map of recently found inconsistencies. If the area where the inconsistency is found is completely uninspected this map is built from scratch.

If the robot is reinvestigating an earlier found inconsistency, or a new inconsistency in a previously investigated space, the "voxel-based map" last created is initially copied to this map. Voxels belonging to this map are labeled empty, occupied, unknown, or conflict. Unknown, means that the voxel has not been labeled yet, empty means that the voxel has been labeled as obstacle free, occupied means that the voxel has been labeled as containing an obstacle, and conflict means that two or more map generation iterations has labeled the voxel as both empty and occupied. The conflicts are resolved as empty if they are interpreted as beam-width conflicts and occupied if they are interpreted as a result of specular reflections. This is described in greater detail in [5]. During the Investigate phase the map generated for navigation by the Initial-Navigation-Map-Generator is replaced by the map generated by the Temporary-Map-Generator.

3. The *Voxel-Based-Map-Generator* creates a long-term map using the Temporary-Map at hand when the Investigate phase is over. The Voxel-Based-Map-Generator assigns a so called "confidence level" to the occupied and empty voxels of this map. The unknown voxels are assigned confidence level 0. This is described in greater detail in Section 9.2.

4. The *Sphere-Based-Map-Generator* creates a Sphere-Based-Map from the Voxel-Based-Map. It assigns spheres around voxels and groups of voxels.

5. The *C-space-Generator* uses the Sphere-Based-Map and templates corresponding to spheres to quickly generate a C-space map using the template method.

The system also contains a path-planner which is incorporated whenever modules like Approach, or higher-level commands fail to bring the robot to certain positions. This path-planner is applied in configuration space and described in Section 7.6.

Another module which is important to the system is the reflexive command filter. The reflexive command filter prevents any of the modules, or higher-level commands from executing commands which would lead to collisions with obstacles described in the last updated C-space map.

An overview of the different modules, as virtual sensor, active component pairs, and the relationship between the modules is given in Figure 9.2. An overview of the system showing the order of activation of the behavioral modules is given in Figure 9.3.

9.2 Confidence Levels for World Exploration Guidance

The system described above will explore the robots workspace and create a world map containing a rough map of all obstacles. The world exploration

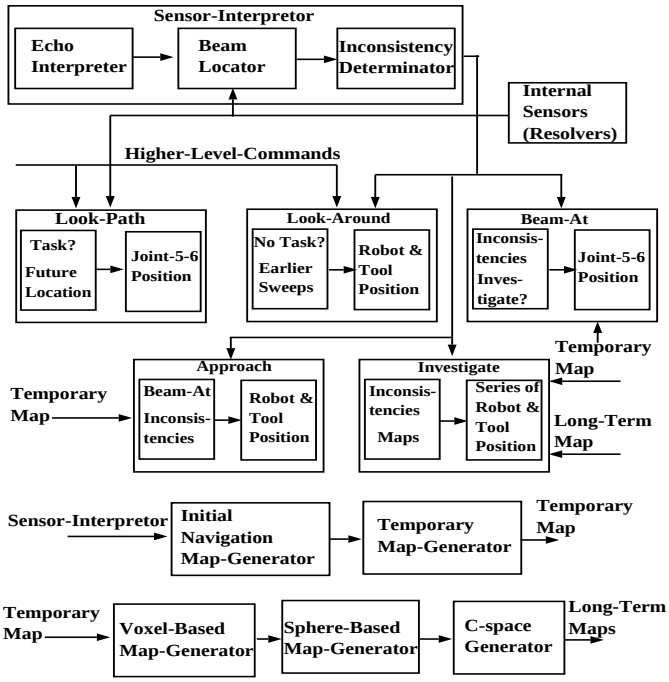


Figure 9.2: The sensor interpreting modules, the world guidance modules, and the map building modules are all connected to each other. The world guidance modules are behavioral modules, and are displayed as virtual sensor and active component pairs.

system will not attempt to reexamine an already mapped obstacle. However, we would like the system to continuously improve the obstacle maps, and also be able to include obstacles introduced into the robot's work space after the world exploration started. To achieve this goal the obstacles in the Voxel-Based-Map are ranked in confidence levels.

The obstacle maps corresponding to the found inconsistencies are ranked in "confidence levels" 0—255. The *confidence level* measures how well a found obstacle has been investigated. If an obstacle has been thoroughly investigated it will have a high confidence level, and its form, size and position will be well

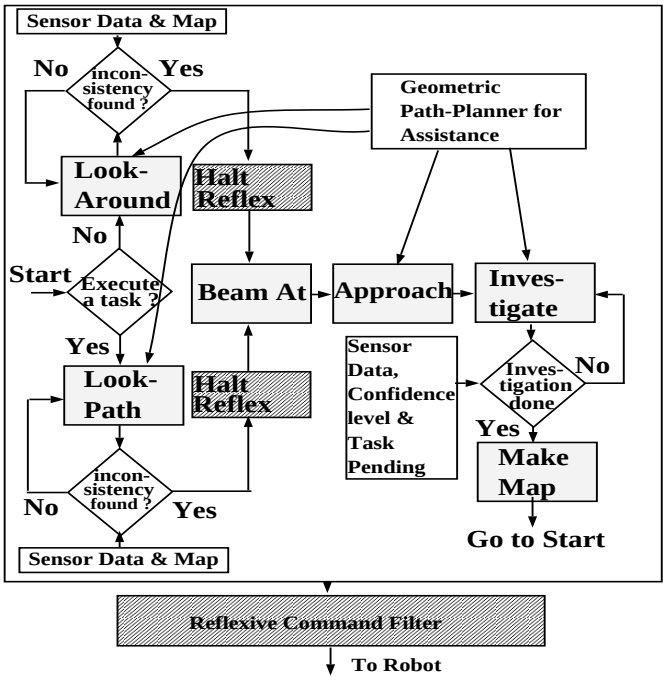


Figure 9.3: An overview of the sonar-based world mapping system. The fixed action patterns are displayed in gray and reflexes in dark gray.

known. If an obstacle has not been thoroughly investigated it will have a low confidence level and its map will be a conservative approximation of the obstacle. The confidence level is increased by subsequent investigations or more thorough investigations.

The use of confidence levels allows the robot to continuously investigate its environment and improve its world map. While the robot is executing a task it only performs minimal world exploration, assigning a low confidence level to the found obstacles, and when the robot is free it will continuously keep perfecting its world map and increase the confidence levels of the found inconsistencies.

A previously found and investigated inconsistency will be placed in the Voxel-Based-Map, assigned a confidence level, and will thereafter not be considered an inconsistency, if its confidence level is higher than the "confidence threshold level". This way the corresponding obstacle will be ignored with respect to further inspection. The *confidence threshold level* is a global confidence level measurement, which corresponds to the overall "trust" in the Voxel-Based-Map. If an obstacle has a confidence level which is equal or larger than the confidence threshold level it will be further ignored with respect to inspection. An obstacle which has a confidence level which is less than the confidence threshold level will be investigated by the system until its confidence level equals the confidence threshold level. An exception from this rule is the case where an obstacle is found during a task execution. In this case the investigation is halted before the confidence level has reached the confidence threshold level, and would therefore easily be resumed when the task is completed.

When all work-space has been investigated the confidence threshold level is raised, and earlier found inconsistencies will be reconsidered for investigation and more thoroughly investigated. Totally unexplored space has a confidence level of 0, and completely known premapped obstacles have a confidence level of 255. In Figure 9.4 an obstacle is considered an inconsistency demanding immediate further investigation if its confidence level is less than 10.

Figure 9.4 shows sonar sensors located at a few different locations. The sonar sensors are displayed as black circles. Obstacles with a confidence level equal or higher than the confidence threshold level will not be considered to be inconsistencies and will be ignored with respect to investigation. Beams registering an inconsistency are shown in gray, and beams registering an obstacle which is not considered to be an inconsistency are displayed in white. When an inconsistency is found, the areas of the work-space where the inconsistency seems to be located are set to a certain confidence level.

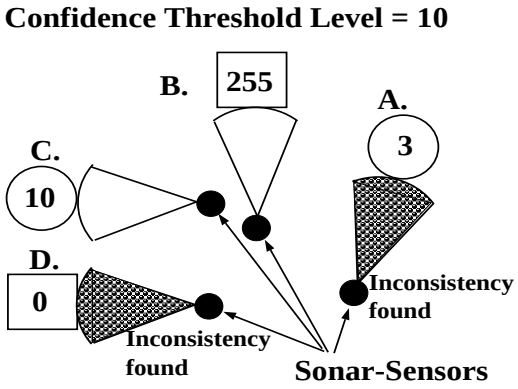


Figure 9.4: Obstacle A was discovered while the robot was completing a task and was therefore only briefly investigated. Obstacle B is a static premapped obstacle, obstacle C an obstacle that has been investigated, and D a recently found obstacle located in unexplored space.

9.3 Conclusions and Overview for Chapter 9

In summary, the sonar-based world mapping system consists of,

1. Sensor interpretation modules, which are used for map generation, and as components of the virtual sensors of the world exploration guidance modules.

2. World exploration guidance modules which control the exploration motion pattern of the system.

3. A Halt-reflex which halts the robot to allow the world exploration guidance modules to properly pre-examine anticipated obstacles in an unknown environment.

4. A geometric path planner which is used to help the world exploration guidance modules to complete motion impaired by known obstacles.

5. A reflexive command filter which protects the robot from collisions with known obstacles.

6. Map generators which take the output from the inconsistency determinator, or "lower level maps" as inputs.

The most central modules of the sonar-based world exploration system are the world exploration guidance modules, which create all of the central motion control for the robot. These modules are, in concept, similar to biological fixed action patterns. Fixed action patterns are low-level behaviors, that generate responses that could be seen as extended reflexive behaviors.

"Investigate" is the most complex of the five world exploration guidance modules. It is inhibited at different confidence levels depending on whether the system has a task to complete or not. The motivation behind this is that, if the robot has a task to perform, it is more important to complete that task than it is to thoroughly explore the inconsistency. In this case the investigation will end at an earlier stage.

When the work-space is totally unexplored and little is known about the obstacles in its environment, we would rather find out something about as many obstacles as possible. In this case the confidence threshold level is low or zero, and the investigation will attempt to cover all obstacles found. Once the confidence threshold level is raised, all obstacles will be reinvestigated so that the overall map can be improved. If entirely new obstacles are found when the confidence threshold level already is high these obstacles will be thoroughly investigated before the rest of the map is improved.

The interaction between the world exploration guidance modules, the confidence threshold level, and the Voxel-Based-Map, creates all the necessary robot motion in sonar-based world mapping system. No central planning is needed. Because of this fact the system is flexible with respect to the conflicting objectives of completing a task, moving safely in unknown space, and continuously updating the work-space map. The system is further robust, and it allows for modular and incremental design.

The system was implemented on the RRC robot mentioned above. The robot systematically investigated obstacles in its environment without having explicit information about how to do this. The exploration was guided by the obstacles and the order in which the obstacles were found. The robot was able to perform tasks while exploring its environment. Due to specular reflections, flat surfaces were sometimes difficult to detect. Thus, this system is not guaranteed to detect all obstacles.

Chapter 10

Stability and Performance of Reflexive and Behavioral Modules

In this chapter I will discuss various concepts regarding stability, convergence, cycling, and performance in the context of reflexive or behavioral modules. Stability and performance analysis of reflex modules and behavioral modules for robotic systems is difficult for several reasons.

1. Robotic systems are very complex. They are multi-dimensional, highly nonlinear, usually unknown to a certain extent, and they often include several complex modules.

2. Reflexive and behavioral modules cannot always be described by a set of differential equations.

3. Common concepts for stability, convergence etc., do not apply. For example, stability in a Lyapunov sense, might not be a desirable feature. We might want the robot to complete a series of tasks before settling down at an equilibrium point, independent of where we start. Further, we might

want the robot to achieve a goal which cannot be expressed as a single robot state, or maybe not even as a set of robot states.

Systems which can be expressed in the following form are referred to as hierarchical systems:

$$\begin{aligned} \dot{\vec{x}}_1(t) &= \vec{f}_1[t, x_1(t)], \\ \dot{\vec{x}}_2(t) &= \vec{f}_2[t, x_1(t), x_2(t)], \\ &\vdots \\ \dot{\vec{x}}_l(t) &= \vec{f}_l[t, x_1(t), \cdots, x_l(t)], \end{aligned} \tag{10.1}$$

An example of a stability theorem which applies to hierarchical systems is the theorem given in [73, 31]. In short this theorem states that if all $\vec{f}_l[t, x_1(t), \cdots, x_l(t)]$ are continuous and bounded, and have continuous and bounded partial derivatives, and all isolated subsystems

$$\dot{\vec{x}}_i = \vec{f}_i[t, \vec{0}, \cdots, \vec{0}, \vec{x}_i(t)]$$

are uniformly asymptotically stable, than the hierarchical system itself is uniformly asymptotically stable. The reason this theorem is not entirely applicable to hierachical behavioral systems is that, it is extremely difficult to express the function of behavioral modules as differential equations. It should also be noted that this theorem is not applicable to systems containing feedback loops.

One of the most general theorems concerning limit cycles is Bendixon's theorem given in [69]. Bendixon's theorem states that for second order autonomous system, i.e. $\dot{x}_1 = f_1(x_1, x_2)$ and $\dot{x}_2 = f_2(x_1, x_2)$, no limit cycle can exist in a region Ω of the phase plane in which $\frac{\delta f_1}{\delta x_1} + \frac{\delta f_2}{\delta x_2}$ does not vanish and does not change sign. This theorem is not useful in the context of robotics and especially behavioral robot architectures. The reason is it only applies to two dimensional systems. It should also be noted that the type of cycling problems that appear in reflexive systems are usually not limit cycles. They might result from quasi-periodic motion (surface in phase-space), hyper-quasi-periodic motion, strange attractors (fractal set in phase space), multiple or time dependent local minimum, or other non-linear or chaotic phenomena. This is described in greater detail in [53].

Despite the lack of methods applicable to complex behavioral systems, analysis is not impossible. A system which is extremely complex, not possible to express as a differential equation, or even unpredictable to a certain degree, can many times still be analyzed with respect to performance. However, it is very difficult to develop a general performance criteria, or an all-purpose analysis, considering that the concept of reflexes and behavioral modules are not well defined in a control sense. Further, what we expect from the different modules might be very different things.

Non-linear control analysis is often applied to special cases of non-linear systems, like:

1. Systems which can be described with explicit non-linear differential equations $\dot{\vec{x}} = \vec{f}(t, \vec{x}(t), \vec{u}(t))$, or $\dot{\vec{x}} = \vec{f}(\vec{x}, \vec{u}) = \vec{f}(\vec{x})$ if the system is autonomous.

2. Systems which can be associated with Lyapunov functions. It should be noted that it often is possible to associate a system with a Lyapunov function without explicit knowledge about the system's differential equations.

3. Systems which can be associated with multiple Lyapunov functions.

4. Systems to which invariant set theorems applies.

5. Systems which can be described as passive or active.

In the same way I will in this chapter describe a few methods for stability and performance analysis of single-module and multiple-module reflexive and behavioral systems. I will not present a general performance measurement method. First I will review common stability concepts which apply to real-time single-module systems. By real-time control systems I refer to systems where time matters. By non-real-time systems I refer to systems for which only the sequence in which events take place matters. I will also discuss real-time multiple-module systems and non-real-time multiple module systems.

To summarize the content of this chapter:

1. In Section 10.1 I will review common stability concepts, and methods, like stability in the sense of Lyapunov, Lyapunov functions, invariant set theorems, hierarchical system theorems, and cycling. I will discuss these methods in the context of reflexive and behavioral systems. In particular, the Local Invariant Set Theorem turns out to be useful in the context of reflexes and behavioral systems.

2. In Section 10.2.1 I will discuss the concept of command-tolerant stability. This concept is introduced to justify analysis of individual modules in a system of hierarchical modules, for which the feedback loop is either non-existent, or can be ignored under certain circumstances. I will show that a system which consists of a module A generating a stable output to a command-tolerant stable module B, will be stable. An example of such a system is shown in Figure 10.4. In the case of linear systems, a system consisting of two stable modules, A and B, which are connected in cascade without feedback, is stable. This result cannot be generalized to arbitrary non-linear systems without the concept of command-tolerant stability.

3. In Section 10.3 I will discuss the use of Lyapunov functions for reflexes and behavioral modules. Instead of using actual robot state vectors, I will use "commanded state vectors", or vectors corresponding to the output from higher-level modules. I will present a stability and convergence analysis

for the reflexive command filter utilizing Lyapunov functions applied in a space spanning the output of higher-level modules. In this section I will also discuss interacting modules, applying multiple Lyapunov functions.

4. In Section 10.4 I will discuss the use of the Local Set-Invariant Theorem for the purpose of reflexes. I will further introduce a concept I refer to as Quasi Lagrange stability. This concept can be seen as an extension of the concept of Lyapunov stability, or a variation of the Local Set-Invariant Theorem. Quasi Lagrange stability is a concept tailored specifically for the reflexes and other systems which purpose is to reach a prescribed region of state-space. A stability and convergence analysis for the flee reflexes is also presented.

5. In Section 10.5 I will introduce a stability concept for non-real time multi-modular interaction, in the case of simple-containment and triggered-containment reflexes. Due to the well-defined characteristics of simple-containment and triggered-containment reflexes, this is an exact concept.

6. In Section 10.6 I will discuss progress measurement functions, which is a very general, but harder to use concept for multi-modular arbitrary behavioral systems.

7. Finally in Section 10.7 I will give a brief overview of the results and conclusions derived in this chapter.

10.1 Common Stability, Cycling, and Convergence Concepts

Equilibrium points are defined as follows [69].

Definition 10.1 *A state $\vec{x}$ is an equilibrium state (or equilibrium point) of the system if once $\vec{x}(t)$ is equal to $\vec{x}$, it remains equal to $\vec{x}$ for all future times.*

Stability (in the Lyapunov sense) is defined as follows [69].

Definition 10.2 *The equilibrium state $\vec{x} = \vec{0}$ is said to be stable if, for any $R > 0$, there exist $r > 0$, such that if $\|\vec{x}(0)\| < r$, then $\|\vec{x}(t)\| < R$ for all $t \geq 0$. Otherwise, the equilibrium point is unstable.*

or in a more compact form :

$$\forall R > 0, \exists r > 0 : \|\vec{x}(0)\| < r \Rightarrow \|\vec{x}(t)\| < R, \forall t \geq 0$$

Instead of picking the equilibrium state $\vec{x} = \vec{0}$ we can pick the equilibrium state to be something arbitrary, $\vec{x} = \vec{x}_{eq}$. In this case Definition 10.2 is written the following way.

Definition 10.3 *The equilibrium state $\vec{x} = \vec{x}_{eq}$ is said to be stable if, for any $R > 0$, there exist $r > 0$, such that if $\|\vec{x}(0) - \vec{x}_{eq}\| < r$, then $\|\vec{x}(t) - \vec{x}_{eq}\| < R$ for all $t \geq 0$. Otherwise, the equilibrium point is unstable.*

or in a more compact form :

$$\forall R > 0, \exists r > 0 : \|\vec{x}(0) - \vec{x}_{eq}\| < r \Rightarrow \|\vec{x}(t) - \vec{x}_{eq}\| < R, \forall t \geq 0$$

It should be noted that in the most general case $\vec{x}(t) = \vec{x}_{eq}(t)$ corrsponds to an equilibrium point. This is, for example, the case in context of error dynamics. I will not consider error dynamics in this thesis. For simplicity, all equilibrium states will be assumed to be $\vec{x} = \vec{0}$ in the remainder of this chapter.

Essentially, stability means that the system trajectory can be kept arbitrarily close to the origin by starting sufficiently close to it. Instability either means that the system "blows up" or cannot be kept within an arbitrarily small volume including the origin.

When using Definition 10.2 in the context of servo-controllers, it is usually assumed that the servo-controller command, or the servo-controller input is static. In other words, when showing stability with respect to an equilibrium point, it is assumed that the input vector $\vec{u}$ in $\dot{\vec{x}}_u = \vec{f}(\vec{x}, \vec{u})$ is time invariant. In fact, for arbitrary time-dependent input vectors $\vec{u}(t)$, stability with respect to an equilibrium point $\vec{0}$ would be an undesirable property. If this were the case the system would always stay close to the equilibrium point no matter what input $\vec{u}(t)$ is given to the system. This is certainly not what we want from a robot servo-controller.

To explicitly state the static input criteria for servo controllers, we could rewrite Definition 10.2 as follows:

Definition 10.4 *The equilibrium state $\vec{x} = \vec{0}$ is said to be stable if,*

$$\forall R > 0, \exists r > 0 : \vec{u}_{servo} = \vec{0}, \|\vec{x}(0)\| < r \Rightarrow \|\vec{x}(t)\| < R, \forall t \geq 0$$

Where $\vec{u}_{servo}$ is the servo-controller input vector.

In other words, we wish the servo-controller to take the robot to a state $\vec{x} = \vec{0}$ (position zero, velocity zero), which also is an equilibrium point. Further, the state $\vec{x} = \vec{0}$ is an equilibrium point for the system under the condition that the desired robot state is $\vec{x} = \vec{0}$. This may seem odd in the context of arbitrary systems. However, it is commonly what is wanted in the context of servo-controllers.

It should be noted that this definition is a special case of Definition 10.1. Sometimes the input state vector to the servo-controller does not correspond to the position where the system comes to rest, and still the system behaves in an intuitively stable manner. Gravity can for example make a servo-controller unable to bring the robot to the desired goal. However, the robot still comes to rest at a point different from the desired position. This system is certainly unstable in a Lyapunov sense if we choose our equilibrium point to be equal to our desired state. However, if we instead chose the final rest point as our

equilibrium point it might be possible to show that our system is stable in a Lyapunov sense. To address this, Definition 10.4 can be generalized as follows:

Definition 10.5 *The equilibrium state $\vec{x} = \vec{0}$ is said to be stable if,*

$$\forall R > 0, \exists r > 0 : \vec{u}_{servo} = \vec{u}_{static}, \|\vec{x}(0)\| < r \Rightarrow \|\vec{x}(t)\| < R, \forall t \geq 0$$

Where $\vec{u}_{servo}$ is the servo-controller input vector, and $\vec{u}_{static}$ is a static vector. The equilibrium point is chosen as $\vec{0}$ for convenience, even though this point might correspond to an undesirable configuration.

The definition for convergence and asymptotic stability is as follows:

Definition 10.6 *A system is* convergent to $\vec{0}$ *if* $\lim_{t \Rightarrow \infty} \vec{x}(t) = \vec{0}$.

Definition 10.7 *An equilibrium point $\vec{0}$ is* asymptotically stable *if it is stable, and if in addition there exists some $r > 0$ such that $\|\vec{x}(0)\| < r$, implies that $\lim_{t \Rightarrow \infty} \vec{x}(t) = \vec{0}$. In other words, an equilibrium point is asymptotically stable if, it is stable and the system is convergent to the equilibrium point.*

With respect to definition 10.7, the region B_r is referred to as the domain of attraction. The region or ball the B_r is defined as follows:

Definition 10.8 *The region B_r or the ball B_r, is the region in state space for which $\|\vec{x}\| < r$.*

Stability with respect to an equilibrium point in the sense of Lyapunov is often proved or discussed using the following theorem given in [69]:

Theorem 10.1 (Local Stability) *If, in the state-space ball B_R, there exists a scalar function $\mathcal{V}$ with continuous first partial derivatives such that*

1. *$\mathcal{V}(\vec{x})$ is positive-definite (locally in B_R)*
2. *$\dot{\mathcal{V}}(\vec{x})$ is negative-semi-definite (locally in B_R).*

then the equilibrium point $\vec{0}$ is locally stable. If, actually, the time derivative $\dot{\mathcal{V}}(\vec{x})$ is locally negative-definite in B_R, then the stability is asymptotic. The scalar function $\mathcal{V}$ is said to be a Lyapunov function.

The proof of this theorem is given in [69]. Figure 10.1 illustrates a Lyapunov function and the corresponding Lyapunov function equipotential curves.

The concept of stability with respect to an equilibrium point is sometimes useful in the context of behavioral systems. When this is the case, we can use Lyapunov functions applied either in the robot's state-space, or to the output of higher-level modules.

Asymptotic stability of a system is also an important property to determine. However, the equilibrium point theorems are often difficult to apply in order to assert this property. The reason is that it often happens that $\dot{\mathcal{V}}$ is only negative-semi-definite.

Another common problem with the concept of equilibrium point stability, in the context of reflexive and behavioral systems, is that it often is that the robot converges to a subset of the state-space rather than a specific point.

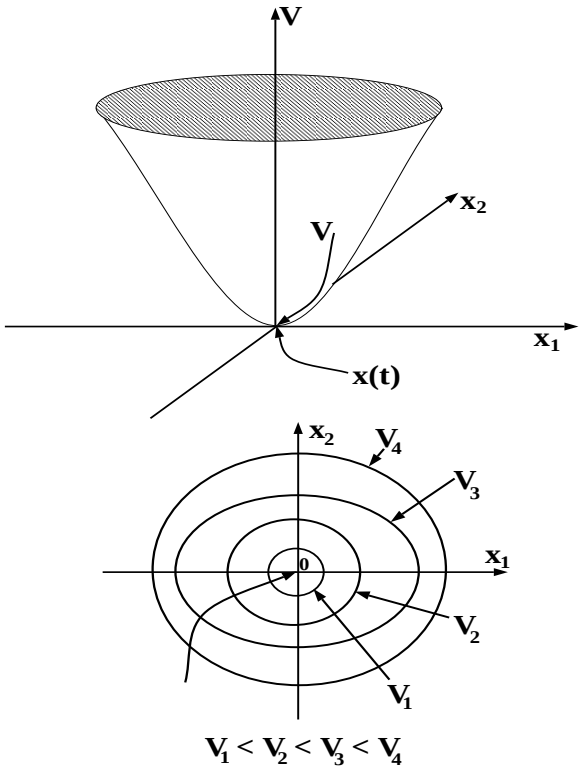


Figure 10.1: This Figure illustrates a Lyapunov function for a two dimensional system. The equipotential curves of the Lyapunov functions are also illustrated.

This is, for example, the case for the flee reflexes. This particular example will be described in greater detail in Section 10.4.

To take care of these kinds of situations we can use *invariant set theorems*, which are a set of powerful theorems attributed to LaSalle. The central concept in these theorems is that of the invariant set, a generalization of the concept of an equilibrium point. In [42] LaSalle refers to stability with respect to a subset of the state-space where $\mathcal{V}$ is constant and thus $\dot{\mathcal{V}} = 0$, as "stability in the sense of Lagrange". Stability in the sense of Lagrange is also discussed in [31].

Definition 10.9 *A set* **G** *is an invariant set for a dynamic system if every system trajectory which starts from a point* **G** *remains in* **G** *for all future time.*

The local invariant set theorem below is given, further explained, and proved in [69]:

Theorem 10.2 Local Invariant Set Theorem *Consider an autonomous system described by* $\dot{\vec{x}} = \vec{f}(\vec{x})$*, with* $\vec{f}$ *continuous, and let* $V(\vec{x})$ *be a scalar function with continuous first partial derivatives. Assume that*

- *for some* $l > 0$*, the region* Ω_l *defined by* $V(\vec{x}) < l$ *is bounded.*
- $\dot{\mathcal{V}}(\vec{x}) \leq 0$ *for all* $\vec{x}$ *in* Ω_l*.*

Let **R** *be the set of all points within* Ω_l *where* $\dot{\mathcal{V}}(\vec{x}) = 0$*, and* **M** *be the union of all invariant sets in* **R***. Then, every solution* $\vec{x}(t)$ *originating in* Ω_l *tends to* **M** *as* $\lim_{t \Rightarrow \infty}$

It should be noted that if **R** is itself invariant (i.e., if once $\dot{\mathcal{V}} = 0$, then $\dot{\mathcal{V}} = 0$ for all future time), then $\mathbf{M} = \mathbf{R}$. The geometrical meaning of this theorem is illustrated in Figure 10.2, where a trajectory starting from within the bounded region Ω_l is seen to converge to the union of all invariant sets **M**. Note that the set **R** is not necessarily connected, nor is the set **M**. Figure 10.3 demonstrates an **M** consisting of one equilibrium point and one limit cycle. In Figure 10.3 a trajectory can enter and leave the set **R**, and converge to the

limit cycle (which is a subset of **R** and **M**). A trajectory can also start inside the cavity, below the limit cycle and converge to the equilbrium point (which is a subset of **R** and **M**).

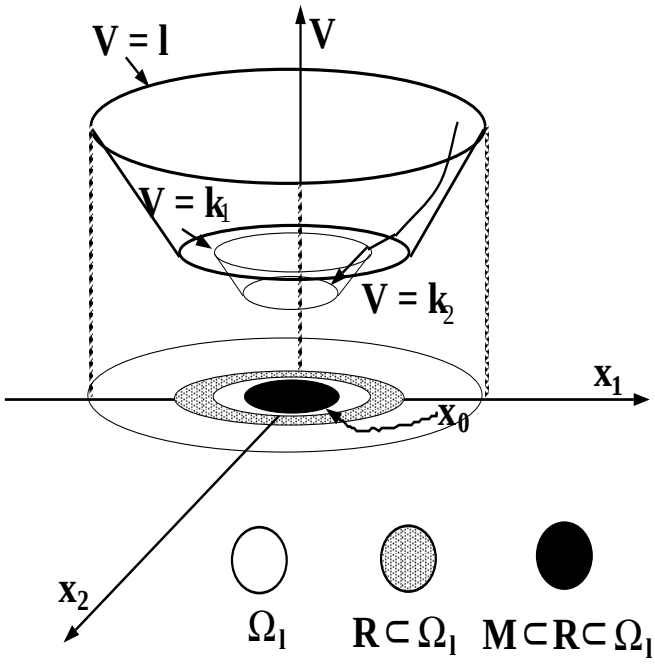


Figure 10.2: Convergence to the union of all invariant sets **M**. $\mathcal{V}$ is constant in the **R** region which means that $\dot{\mathcal{V}} = 0$ for arbitrary trajectories. If $\mathcal{V}$ is not constant everywhere in **R**, it must be the case that only certain trajectories within **R** for which $\mathcal{V}$ is constant are possible.

A partial proof of this theorem is given in [69], and consists of two steps. The first part of the proof is to show that $\dot{\mathcal{V}}$ goes to zero, in other words R. The second part of the proof is to show that any bounded trajectory within R must converge to an invariant set. This invariant set might be an equilibrium point, among many equilibrium points, a limit cycle, etc..

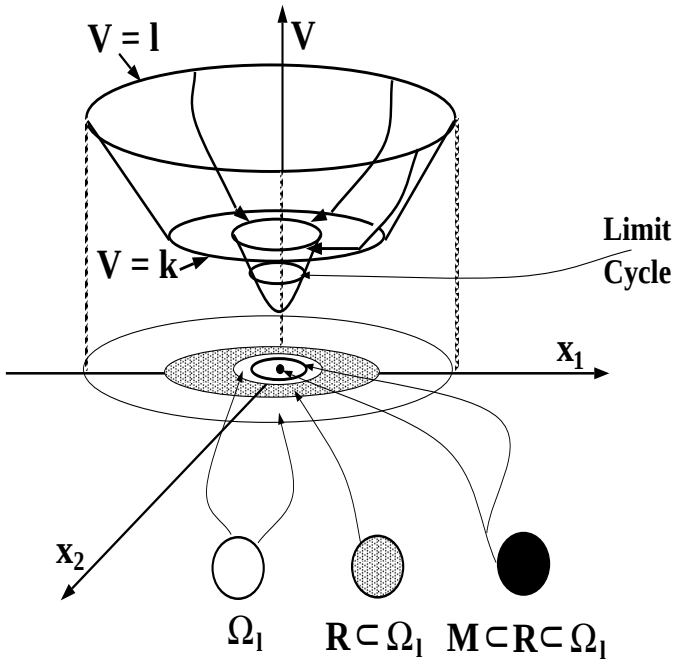


Figure 10.3: In the case illustrated here the paths will take the robot across the region **R** and finally to the cavity inside **R**. In this cavity $\dot{\mathcal{V}} < 0$, except at the limit cycle and the minimum in the center.

It should be noted that the asymptotic stability result in the local Lyapunov theorem can be viewed as a special case of the above invariant set theorem, where the set **M** consists only of the origin.

This concept is quite useful in, for example, in the case of the flee reflexes. Consider the system consisting of the reflexive command filter, and the wall-emulating flee reflex. The reflexive command filter guarantees, under static conditions (the wall not approaching), that the robot will stay within the free space if it already is there. For this reason the free-space corresponds to our invariant set. Further, if the robot is intersecting a wall, the flee reflex will drive the robot into the invariant set (free-space). To show this, we can use the

flee potential as our function $\mathcal{V}$. When the flee reflex guarantees that $\dot{\mathcal{V}} = 0$ in free space and $\dot{\mathcal{V}} < 0$ elsewhere, all trajectories converge to the invariant set free-space according to the local invariant set theorem. In the case of the block-emulating flee reflex, the invariant set consists of the intersection of the free-space and the space where the flee potential is zero. It should be noted that the region where the flee potential is zero is a subset of the free-space. These two cases will be discussed in greater detail in Section 10.4.1.

It should be noted that in some cases the desired goal state for an individual module cannot be expressed as a subset of either the state-space or as a subset of possible higher-level module output vectors. Further, the concept of stability and convergence might in some cases not even have an intuitive meaning, nor be what we desire from the system. To handle these kinds of situations we have to resort to other types of performance measurement methods. We also need to distinguish between "real time" control applications, and purely logical performance measurement methods.

In the remainder of this chapter I will discuss Lyapunov functions applied to the output of higher-level modules, the use of the invariant set theorem and similar definitions, and finally introduce new specialized performance measurements. I will also discuss stability, convergence, and performance measurement methods in the context of interacting modules.

10.2 The Concept of Command-Tolerant and Monotonic Stability.

In this section the concepts of "command-tolerant stability" and "monotonic-in/monotonic-out stability" is introduced. The purpose behind introducing the concept of "command-tolerant stability" is to validate the use of Lyapunov functions for the output vectors of higher-level modules, which will be described in Section 10.3. The concept of "monotonic-in/monotonic-out stability" is a more specific but closely related concept. It is introduced for the purpose of analysis of the reflexive command filter and other similar reflex modules.

10.2.1 The Concept of Command-Tolerant Stability.

A system consists of two modules, A and B, which are connected in cascade. Module A generates an output which is stable with respect to the equilibrium point $\vec{x}_{A,eq}$ (in Lyapunov sense). Module B is the servo-controller/robot module. The output of module A ($\vec{x}_A$) is the input to module B ($\vec{u}_B$). The resulting system is illustrated in Figure 10.4.

If module B also generates a stable output, assuming a fixed input $\vec{u}_{B,fixed}$, it is natural to assume that the system is stable. For real systems with practical servo-controllers this is true. An example of this situation would be

the following. An operator A gives the robot system B a position command $\vec{x}_A = \vec{u}_B = \vec{0}$. The robot converges in a stable manner to the position $\vec{x}_B = \vec{0}$. In other words, module B (the robot system) is stable assuming a fixed command $\vec{u}_B = \vec{0}$. A higher-level system A' gives the robot system B a command which corresponds to a trajectory converging to $\vec{0}$ in a stable manner. In this situation we still expect the robot system to converge to $\vec{x}_B = \vec{0}$ in a stable manner. This will be referred to as "command-tolerant stability". If the robot system does not converge in the latter case, module B is not command-tolerant stable.

With the output from module A "behaving stable" with respect to the equilibrium point $\vec{x}_{A,eq}$ is meant that the output signal will always stay within $\|\vec{x}_A(t) - \vec{x}_{A,eq}\| < R$, where R can be chosen arbitrarily, if the output signal initially is less than r ($\|\vec{x}_A(0) - \vec{x}_{A,eq}\| < r$.

Definition 10.10 *The equilibrium state $\vec{x} = \vec{0}$ is said to be* command-tolerant stable *if, for any $R > 0$, there exists an $\epsilon > 0$, and there exists an $r > 0$, such that if $\|\vec{u}(t) - \vec{u}_{eq}\| < \epsilon$ for all $t \geq 0$, where $\vec{u}_{eq}$ is the desired input vector, and $\|\vec{x}(0)\| < r$, then $\|\vec{x}(t)\| < R$ for all $t \geq 0$. or in a more compact form,*

$$\forall R > 0, \exists \epsilon > 0, \exists r > 0 : (\|\vec{u}(t) - \vec{u}_{eq}\| < \epsilon, \forall t \geq 0), \|\vec{x}(0)\| < r \Rightarrow \|\vec{x}(t)\| < R, \forall t \geq 0$$

With respect to the example, command-tolerant stability means that, if the input $\vec{u}_B$ is stable with respect to the equilibrium point $\vec{u}_{eq}$, then the output

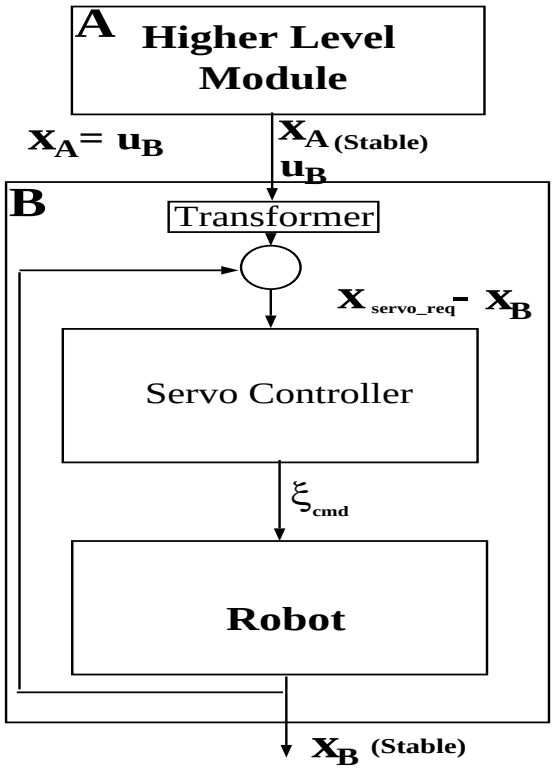


Figure 10.4: Module A is generating an output to the servo-controller which is stable with respect to an equilibrium point. Module B, which consists of the robot and the servo-controller, generates a stable output if either given a static input, or a variable input which is stable with respect to an equilibrium point.

$\vec{x}_B$ is stable with respect to the equilibrium point $\vec{0}$. This should be seen in contrast to assuming a static input when showing stability for a module. This is a very reasonable criterion for a stable servo-controller. In fact a servo-controller which is not command-tolerant stable is useless for robotic systems. It should be noted that both Definition 10.10 and 10.4 are special cases of Definition 10.2.

It should be noted that $\vec{u}_B$, and $\vec{u}_{eq}$ might be vectors which correspond to

desired robot state vectors (e.g. they are velocity and position commands). This is commonly the case for servo-controllers. However, this does not have to be the case. For example, if the servo-controller includes a trajectory generator, $\vec{u}_B$ could correspond to a tool position in the work space.

Using Definition 10.10 allows us to analyze modules in a hierarchical system individually, and draw conclusions about the stability of the entire system, as long as there are no feedback loops, or the feedback loops can be ignored[1].

What we intend to show in this section is that a system which consists of the following two modules,

- A, which generates an output which behaves stable with respect to an equilibrium point $\vec{u}_{eq}$, and
- B, which is command-tolerant stable with respect to the equilibrium point $\vec{0}$.

is stable with respect to the equilibrium point $\vec{0}$[2].

Theorem 10.3 *If the higher-level control output, or in other words, the servo-controller input vector, can be shown to be stable in a Lyapunov sense, and the servo-controller/robot module is command-tolerant stable, then the entire system is stable in a Lyapunov sense.*

[1] This is a generalization of the fact that, for the special case of linear systems, two stable modules in cascade constitutes a Lyapunov stable system.

[2] In a Lyapunov sense

This is true by definition. However, it is possible to present a proof which is based on the formal definition.

Proof Assume the following:

1. For any $\alpha > 0$, there exists $\gamma > 0$, such that if $\|\vec{u}(0) - \vec{u}_{eq}\| < \gamma$, then $\|\vec{u}(t) - \vec{u}_{eq}\| < \alpha$ for all $t \geq 0$. In other words assume the higher-level module generates a stable input to the servo controller.

2. The servo controller is command-tolerant stable, i.e.,

$$\forall R > 0, \exists \epsilon > 0, \exists r > 0 : (\|\vec{u}(t) - \vec{u}_{eq}\| < \epsilon, \forall t \geq 0), \|\vec{x}(0)\| < r \Rightarrow \|\vec{x}(t)\| < R, \forall t \geq 0$$

When ϵ is defined by modules external to the servo controller, it is the only entity in the definition for command stability which cannot be picked arbitrarily. However, from (1) we can pick $\alpha > 0$ arbitrarily, particularly the $\alpha = \epsilon$ required for (2) to be fulfilled. Assuming we always can pick the required $\epsilon > 0$, (2) can be rewritten as,

$$\forall R > 0, \exists r > 0 : \|\vec{x}(0)\| < r \Rightarrow \|\vec{x}(t)\| < R, \forall t \geq 0$$

and thus we have shown that, assuming the servo-controller input $\vec{u}(t)$ acts stable, the entire system is stable. □

Observe that the command-tolerant stability theorem can be used in cascade, so that if all modules can be assumed to be command-tolerant stable, we only need to show the stability of the input to the highest-level module.

For the robotic systems considered here, the input vector usually consists of a set of position commands and a zero velocity vector. In these cases, it is only necessary to show that the position commands generated by higher-level controls to the servo-controller are stable.

The analysis of the reflexive systems with respect to convergence and asymptotic stability becomes considerably easier if we use the concept of a *command-tolerant convergent* module and assume a "command-tolerant asymptotically stable" servo-controller. I will define "command-tolerant convergence" as follows :

Definition 10.11 *A module is* locally command-tolerant convergent *if there exists some* $r > 0$ *such that* $\|\vec{x}(0)\| < r$, *and* $\lim_{t \Rightarrow \infty} \vec{u}(t) = \vec{u}_{eq}$, *implies that* $\lim_{t \Rightarrow \infty} \vec{x}(t) = \vec{0}$

I further will define "local command-tolerant asymptotic stability" as follows :

Definition 10.12 *An equilibrium point* $\vec{0}$ *is* locally command-tolerant asymptotically stable *if it is command-tolerant stable, and if in addition there exists some* $r > 0$ *such that* $\|\vec{x}(0)\| < r$, *and* $\lim_{t \Rightarrow \infty} \vec{u} = \vec{u}_{eq}$, *implies that* $\lim_{t \Rightarrow \infty} \vec{x}(t) = \vec{0}$. *In other words, an equilibrium point is command-tolerant asymptotically stable if it is command-tolerant stable and command-tolerant convergent.*

10.2.2 The Concept of Monotonic-In/Monotonic-Out Stability.

That a system output or state vector, or an arbitrary signal is monotonically approaching a point is defined as follows:

Definition 10.13 *A system output/input or a signal* $\vec{x}(t)$ monotonically approaches *an point* $\vec{x}_{eq}$ *if* $\|\vec{x}(t) - \vec{x}_{eq}\| \leq \|\vec{x}(t - \delta t) - \vec{x}_{eq}\|, \forall t \geq 0, \delta t > 0$.

The point $\vec{x}_{eq}$ could be an equilibrium point for the system, in other words once $\vec{x}(t) = \vec{x}_{eq}$ it will remain there.

Definition 10.14 *An equilibrium point* $\vec{x}_{eq}$ *is said to be monotonically stable in the region* Ω *if the corresponding system output or state space vector at all times monotonically approaches the equilibrium point everywhere in the region* Ω.

It should be noted that monotonic stability also means stability in the sense of Lyapunov.

Monotonic-In/Monotonic-out stability is defined as follows:

Definition 10.15 *The equilibrium point* $\vec{0}$ *is said to be* monotonic-in/monotonic-out stable *if the fact that the input vector* $\|\vec{u}(t)\|$ *is monotonically approaching a point* $\vec{u}_{eq}$*, implies that the output state vector monotonically approaches the equilibrium point* $\vec{0}$*. In other words if,*

$$\|\vec{u}(t)-\vec{u}_{eq}\| \leq \|\vec{u}(t-\delta t)-\vec{u}_{eq}\|, \forall t \geq 0, \forall \delta t > 0 \Rightarrow \|\vec{x}(t)\| \leq \|\vec{x}(t-\delta t)\| \forall t \geq 0, \forall \delta t > 0$$

then the system is monotonic-in/monotonic-out stable.

It should be noted that monotonic-in/monotonic-out stability is a very strict demand which might be hard to prove. However, in section 10.3.1 it is shown that the reflexive command filter is monotonic-in/monotonic-out stable. Further, what is referred to as a non-overshooting servo-controller in sections 5.4 and 10.3.1 is another example of a monotonic-in/monotonic-out stable module[3].

A non-overshooting servo-controller is a servo-controller, which for all position commands, generates paths which monotonically converge towards the desired joint positions, if given from a zero initial velocity. In other words, assuming that the servo-controller-robot module has settled, a new monotonically converging input will result in the servo-controller-robot module monotonically converging to the new goal. A non-overshooting servo-controller will "overshoot" if, for example, it receives a position command located in the opposite direction of where it it is going. Examples of non-overshooting servo-controllers are soft (over- or critically-damped) PD-controllers, bang-bang, and sliding-mode-controllers.

The following theorem regarding monotonic stability is very useful for example, in the case of the reflexive command filter. The theorem refers to Figure 10.5.

[3]At least in the way they are used in the context of reflexive obstacle avoidance

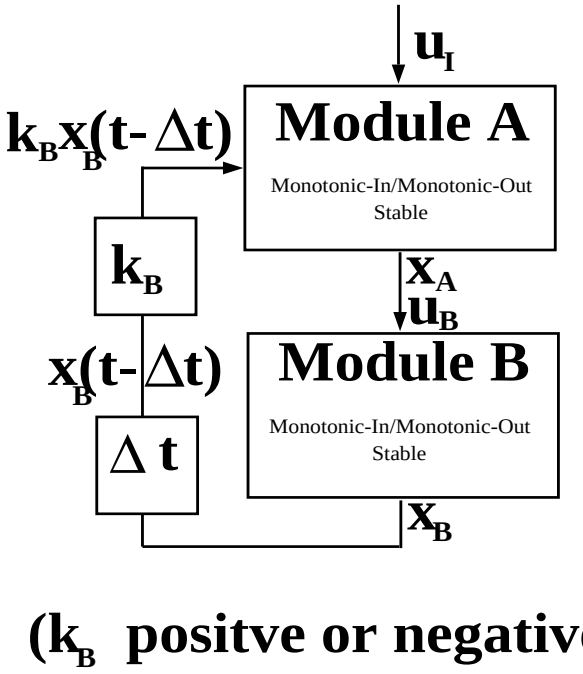


Figure 10.5: Two monotonic-in/monotonic-out stable modules connected via a feedback loop.

Theorem 10.4 *A system consists of two modules A and B connected to each other via a feedback loop with an arbitrary time-delay* $\Delta t > 0$. *If,*

- *All signals in the system are either continuous, or have a finite number of discontinuities.*
- *module A is monotonic-in/monotonic-out stable,*
- *module B is monotonic-in/monotonic-out stable,*
- *the input to A corresponding to the feedback loop is considered a part of the input vector to the module A,*
- *and any additional input to the module A is either constant, or monotonically approaching a certain input state,*

the system consisting of module A and B, including the feedback loop is monotonic-in/monotonic-out stable.

Proof With respect to Figure 10.5 the input vector to A is $\vec{u}_A(t) = (k_B\vec{x}_B(t-\Delta t), \vec{u}_I(t))$, where k_B can be a negative or positive non-zero constant. The point the input is supposed to approach in this case is $(\vec{0}, \vec{u}_{I,eq})$. Assume that the system consisting of module A and B, including the feedback loop is not monotonic-in/monotonic-out stable. This means that $\|\vec{x}_B(t)\|$ the normal of the system state vector at some time t is increasing. Assume that $\|\vec{x}_B(t)\|$ increases for the first time at a time $t \geq \Delta t$. This means that at some time $t \geq \Delta t$ and for some $\delta t < \Delta t$, $\|\vec{u}_B(t-\delta t)\| > \|\vec{u}_B(t)\|$. That this is true for some $\delta t < \Delta t$ follows from the fact that if, all signals are continuous, or have a finite number of discontinuities, then $\|\vec{u}_B(t-\delta t)\| > \|\vec{u}_B(t)\|$ for some t and δt implies that $\|\vec{u}_B(t_2-\delta t_2)\| > \|\vec{u}_B(t_2)\|$ for some t_2 and δt_2, where δt_2 can be chosen arbitrarily small.

Module B is monotonic-in/monotonic-out stable which means that if, $\|\vec{u}_B(t-\delta t)\| > \|\vec{u}_B(t)\|$ then $\|\vec{u}_A(t^{'}-\delta t^{'})\| > \|\vec{u}_A(t^{'})\|$ for some $t^{'} \leq t$ and some $\delta t^{'} < \Delta t$. However, module A is also monotonic-in/monotonic-out stable which means that if, $\|\vec{u}_A(t^{'}-\delta t^{'})\| > \|\vec{u}_A(t^{'})\|$ then $\|k_B(\vec{x}_B(t^{''}-\Delta t), \vec{u}_I(t^{''}))\| > \|k_B(\vec{x}_B(t^{''}-\Delta t-\delta t^{''}), \vec{u}_I(t^{''}-\delta t^{''}))\|$ for some $t^{''} \leq t^{'}$ and some $\delta t^{''} < \Delta t$. In other words $\|\vec{x}_B(t^{''}-\Delta t)\| > \|\vec{x}_B(t^{''}-\Delta t-\delta t^{''})\|$ for some $t^{''} \leq t^{'}$ and some

$\delta t'' < \Delta t$. This follows from the fact that $\vec{u}_I(t))$ is assumed to be monotonically approaching $\vec{u}_{I,eq}$, and $k_B\vec{x}_B(t-\Delta t)$ and thus $\vec{x}_B(t-\Delta t)$ is monotonically approaching $\vec{0}$. However, $\|\vec{x}_B(t'' - \Delta t)\| > \|\vec{x}_B(t'' - \Delta t - \delta t'')\|$, $t'' \leq t' \leq t$ where $\delta t'' < \Delta t$ means that the original assumption that $\|\vec{x}_B(t)\|$ increases for the first time at a time $t \geq \Delta t$ is violated.

Now assume that $\|\vec{x}_B(t)\|$ increases for the first time at a time $t < \Delta t$. Initially assume that $\vec{u}_A(0) = (k_B\vec{x}_B(0), \vec{u}_I(0))$ where $\vec{x}_B(0) = \vec{x}_B(0)$. For all times $t < \Delta t$ the input to module A $\vec{u}_A(t)$ is $(k_B\vec{x}_B(0), \vec{u}_I(t))$ which is a signal which monotonically approches $(\vec{0}, \vec{u}_{I,eq})$. In other words, it cannot be true that $\|\vec{x}_B(t)\|$ increases for the first time at a time $t < \Delta t$, which means that it never increases. In other words the entire system is monotonically stable. □

In the remainder of this chapter I will discuss stability, convergence and cycling problems with respect to the position commands generated by the reflexive and behavioral modules, and thus avoid dealing with the true state vector including both the actual position and the actual velocity. When using the command-tolerant stability theorem to show overall system stability, we must be able to show that higher-level modules generate stable outputs.

However, whether the higher-level modules generate stable outputs or not depends on the robot state. In other words the command-tolerant stability theorem is not a method to entirely decouple modules in a hierarchical system. Why it still is useful is due to the fact that we usually can predict the behavior

of higher-level behavioral modules without having precise knowledge about the robot state vector. However, we usually have to assume that the robot state vector is in a certain region of state-space to be able to predict the behavior of the behavioral module. If we can do this, the modules in a hierarchical system can be decoupled from each other using the command-tolerant stability theorem.

10.3 The Use of Lyapunov Functions For Higher-Level Modules

Stability analysis is commonly applied to servo-controllers. However, we would like to extend stability analysis to the outputs of higher-level modules as well. Examples of higher-level outputs are commanded state vectors (configurations and velocities), sensor states, recorded data, or internal map states. To obtain noted conclusions with respect to higher-level modules, the corresponding Lyapunov functions must be chosen to reflect the progress in the output of the module.

The inclusion of sensor data, map data, and other types of data in the state vector, might allow what looks like cycling with respect to commanded joint coordinates and velocities, although the system is making progress with respect to higher-level goals. In this context it is important to choose a Lyapunov

function which depends on the entire output vector from the module.

An example is the following Lyapunov function for the reflexive command filter: $\mathcal{V}_{ref-filt}(\vec{x}_s) = \frac{1}{2}\sum_{i=1}^{N} K_{x,ii}(x_{des,i} - x_{s,i})^2$ where $\vec{x}_s$ is the approved configuration generated by the reflexive command filter, and $\vec{x}_{des}$ is the goal configuration. It should be noted that the reflexive command filter attempts to minimize the joint energies individually, which in our 4-D case can be expressed as the vector function: $\vec{\mathcal{J}}_{ref-filt}(\vec{x}_s) = \frac{1}{2}[K_1(x_{des,1} - x_{s,1})^2, K_2(x_{des,2} - x_{s,2})^2, K_3(x_{des,3} - x_{s,3})^2, K_4(x_{des,4} - x_{s,4})^2]$. The function corresponding to this stricter energy criterion can be used to form a Lyapunov scalar function according to $\mathcal{V}_{ref-filt}(x_{s,i}) = \mathcal{J}_{1,ref-filt}(x_{1,s}) + \mathcal{J}_{2,ref-filt}(x_{2,s}) + \mathcal{J}_{3,ref-filt}(x_{3,s}) + \mathcal{J}_{4,ref-filt}(x_{4,s})$. Observe that these two Lyapunov scalar functions are the same.

Another example of a Lyapunov function applied to a higher-level module is the "energy measurement" given for the guiding potential functions given in Section 7.2. For the *configuration-space guiding potential function*, the Lyapunov function would be: $\mathcal{V}_{guid-pot}(\vec{x}_s) = \frac{1}{2}\sum_{i=1}^{N} K_{x,i}(x_{des,i} - x_{s,i})^2$, in other words the same as for the reflexive command filter. For the *augmented task-space potential function* the Lyapunov function would be: $\mathcal{V}_{guid-pot}(\vec{x}_s) = \frac{1}{2}\sum_{i=1}^{N} K_{y,i}(y_{des,i} - y_{s,i})^2$ where $\vec{y}_{des}$ is an N-dimensional augmented task-space goal and $\vec{y}_s$ an N-dimensional augmented task-space command output from the guiding potential function.

In Section 10.3.1 it will be shown that the Lyapunov function of the reflexive command filter is positive-definite, and further its time derivate: $\dot{\mathcal{V}}_{ref-filt}(\vec{x}_s) = -\frac{1}{2}(K_1(\dot{x}_{s,1}(x_{des,1} - x_{s,1})) - K_2(\dot{x}_{s,2}(x_{des,2} - x_{s,2})) - K_3(\dot{x}_{s,3}(x_{des,3} - x_{s,3})) - K_4(\dot{x}_{s,4}(x_{des,4} - x_{s,4})))$ is negative-definite or negative-semi-definite if the "desired" joint velocities generated by the reflexive command filter $\dot{\vec{x}}_s$ are either directed towards the joint goal, or zero.

It is also true that the Lyapunov function for the configuration-space guiding potential function: $\mathcal{G}_{guid-pot}(\vec{x}_s) = \frac{1}{2}\sum_{i=1}^{N} K_{x,i}(x_{des,i} - x_{s,i})^2$ is positive-definite, and further the time derivate: $\dot{\mathcal{G}}_{guid-pot}(x_s) = -\frac{1}{2}\sum_{i=1}^{N} K_{x,i}\dot{x}_{s,i}(x_{des,i} - x_{s,i})$ is negative-definite or semi-definite if, $\sum_{i=1}^{N} K_{x,i}\dot{x}_{s,i}(x_{des,i} - x_{s,i}) \geq 0$.

In the case of the block-emulating reflex, it is possible to define a Lyapunov function for the startle and the retraction reflexes, due to the fact that the trajectories are predetermined in each case. In these cases, the robot has to reach a certain point, the equilibrium point, which is defined by the flee potentials and the robot states at the time the flee reflex is turned on. For the startle reflex the equilibrium point can be computed as $\vec{\theta_{eq}} = -k_{\omega_i}\int_{t_0}^{t_{off}} J^T \nabla \mathcal{F}_{flee}(x, y, z)dt$, where $\vec{\theta_{eq}}$ is the equilibrium point, $\mathcal{F}_{flee}$ the flee potential, t_0 the time the startle reflex is initialized and t_{off} the time the startle reflex is turned off. When J^T and $\mathcal{F}_{flee}(x, y, z)$ depend on $\vec{\theta}(t)$ and $(x(t), y(t), z(t))$ respectively, and t_{off} depends on $\mathcal{F}_{flee}(x, y, z)$ and $\vec{\theta(t)}$ this computation usually has to be done numerically. For the retraction reflex, the equilibrium point would be

$\vec{\theta_{eq}} = \int_{t_0}^{t_{off}} J_A^{-1} k_v \nabla \mathcal{F}_{flee}(x, y, z, \phi) dt$. Once the equilibrium points are known, the Lyapunov function can be defined as a distance measurement to the equilibrium point. However, it should be noted that this is an awkward way of analyzing the flee reflexes. The invariant set theorems are superior in this case.

10.3.1 Stability and Convergence Analysis for the Reflexive Command Filter

It was shown in Chapter 5 that the reflexive command filter guarantees that the robot will stay within a certain approved free-space prism. This is indeed the primary objective of the reflexive command filter. However, this does not guarantee that the system is stable. It is possible for the robot to be caught in a limit cycle and still never leave the free-space prism. Assuming a stable (command-tolerant stable) servo-controller and robot system and a position-based reflexive command filter relying on a non-overshooting servo-controller, it is also possible to show stability using Lyapunov functions. Or better, assuming a command-tolerant uniformly asymptotically stable servo-controller and robot system it can be shown the combined servo-controller, robot and reflexive command filter system is command-tolerant uniformly asymptotically stable, by showing that the reflexive command filter is command-tolerant uniformly asymptotically stable. In this section I will first give a thorough, but tedious proof of the fact that the system consisting of the reflexive command

filter, a non-overshooting servo-controller/robot system, including the feedback loop to the reflexive command filter is stable. Next, I will explain why the system is command-tolerant uniformly asymptotically stable.

An overview of the system anlyzed here is given in Figure 10.6. The feedback loop from the robot state to the C-space inspector is used by the C-space inspector to guarantee that the approved free-space prism contains the current position. In the case of a position-based reflexive command filter, the feedback loop from the robot state to the active command filter cannot affect the performance of the system. The reason for this is that the approved position command is a fixed point within the free-space prism which is independent of the current robot state vector.

To show stability for the system shown in Figure 10.6 it is not necessary to analyze the entire system consisting of the robot, the servo-controller, the reflexive command filter, and all the feedback loops. The servo-controlled robot system is assumed to be stable by itself, and the only feedback loop we need to worry about is the feedback loop from the robot state to the Cspace inspector.

The Lyapunov function in the case of the position-based reflexive command filter is $\mathcal{L}_{ref-filt}(x_s) = \frac{1}{2}\sum_{i=1}^{N} K_{x,i}(x_{des,i} - x_{s,i})^2$, where $\vec{x}_s$ is the command output from the reflexive filter, and the commanded input to the servo-controller. $\vec{x}_{des}$ is the desired goal position, and the equilibrium point in this context.

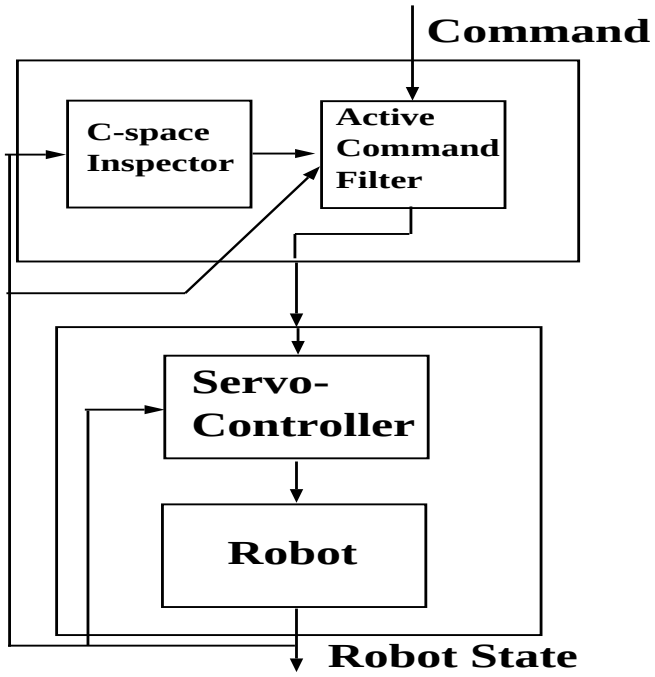


Figure 10.6: Overview of a system consisting of a reflexive command filter, a servo controller and the robot.

We need to show that,

1. $\mathcal{L}_{ref-filt}(\vec{x}_s)$ is continuous, has continuous first partial derivatives, and is positive-definite.

2. $\dot{\mathcal{L}}_{ref-filt}(\vec{x}_s)$ is negative-semi-definite.

independent of the feedback loop to the C-space inspector.

It is evident that $\mathcal{L}_{ref-filt}(\vec{x}_s)$ is continuous, has continuous first partial derivatives, and is positive-definite. The challenge lies in showing that $\dot{\mathcal{L}}_{ref-filt}(\vec{x}_s)$ is negative-semi-definite. To do this we need to examine the way the approved position commands are generated by the reflexive command filter.

If the position request $\vec{x}_{des}$ is located inside the approved free-space prism,

it is approved. As a result, the output of the reflexive command filter would remain constant at $\vec{x}_s = \vec{x}_{des}$ Correspondingly, $\dot{\mathcal{L}}_{ref-filt}(\vec{x}_s) = 0$. This is true independent of the current robot position, and thus true independent of the feedback loop to the C-space inspector.

After the next execution cycle of the C-space inspector, the approved free-space prism will be updated so that the position request either will lie within the approved free-space prism, or at a fixed location at the face of the approved free-space prism. When the front-face of the approved free-space prism is either identical to the front-face of the previous approved free-space prism, or updated so that it is closer to the position request, then the approved positions monotonically approach the requested position. In other words, $\dot{\mathcal{L}}_{ref-filt}(\vec{x}_s) \leq 0$. This is also true independent of the current position, and thus true independent of the feedback loop to the C-space inspector. It is evident that, even though the feedback loop to the C-space inspector determines the size of the free-space prisms, this feedback loop can not influence the stability of the output from the reflexive command filter, assuming a non-overshooting servo controller and, assuming that the reflexive command filter is functional.

Why the corner point closest to the position request in subsequent approved free-space prisms monotonically approaches the position request has to do with the way the C-space inspector searches C-space and generates free-space

prisms. The search begins at the borders of the previously approved free-space prism, and is expanded into the new requested prism in a wave-like manner, towards the requested position, as illustrated in Figure 5.5. This means that if no obstacles are found, the corner point closest to the requested position will get closer to the requested position. On the other hand if an obstacle is found, the corner point will either remain the same or be updated so that it gets closer to the position request along at least one joint.

To summarize these results, the approved position is either identical to the requested position or on the face of the approved free space prism, and therefore monotonically converges to the position request. This way both criteria necessary for stability are fulfilled. Assuming a command-tolerant stable servo-controller/robot pair, this means that the entire system is stable. It should be noted that if there are obstacles in C-space we cannot guarantee asymptotic stability when these obstacle could halt further progress.

In Section 10.2.1 it was shown that if two modules A and B are connected to each other via a feedback loop with an arbitrary time-delay $\Delta t > 0$, and they both are monotonic-in/monotonic-out stable, the combined system is monotonic-in/monotonic-out stable.

The reflexive command filter can be viewed as the module B with the inputs:

1. The incoming desired position.

2. The current position.

Assuming that,

1. The desired position is fixed,

2. The current position is monotonically approaching a reference point (the fixed desired position),

the reflexive command filter will generate a position command which is either identical to the previous command, or is closer to the desired position than the previous command was. This means that the reflexive command filter is monotonic-in/monotonic-out stable. Assuming a monotonic-in/monotonic-out stable servo-controller/robot system this means that the combined system also is monotonic-in/monotonic-out stable, according to theorem 10.4.

10.3.2 The Use of Lyapunov Functions in the Case of Interacting Modules

If we are using the configuration-space guiding potential function in conjunction with the reflexive command filter the combined system is guaranteed to be stable in the sense of Lyapunov, if both modules are stable in the sense of Lyapunov. The reason is that the same Lyapunov function can be used for both the reflexive command filter and the configuration-space based guiding potential function. That means that if the time derivative Lyapunov function

for the reflexive filter is negative-semi-definite, it is negative-semi-definite for the guiding potential function. Even though the modules generate different trajectories, these trajectories will always decrease the Lyapunov function for both modules. No matter how much we switch among the modules, we are always guaranteed stability. Further, if both modules monotonically decrease the Lyapunov function (negative-definite), we are guaranteed convergence independent of which module is in control at any given time.

However, this is not true if, for example, we are using the augmented task-space guiding potential function, or the work-space guiding potential function in conjunction with the reflexive command filter. In this case $\dot{\mathcal{V}}_{guid-pot}(t)$ corresponding to the guiding potential function might be positive, while $\dot{\mathcal{V}}_{ref-filt}(t)$ corresponding to the reflexive filter is negative. This is a source of limit cycling when it allows for states to be visited multiple times when switching control modes.

If we are switching among modes of control which cannot be described with the same Lyapunov function, we need to make sure that this switching does not cause cycling. We may switch among a finite number of modes of control associated with different Lyapunov functions[4], and still guarantee no sustained limit cycling, and stability provided:

Requirement 10.1

[4] Associated with different Lyapunov functions not by accident, but with necessity.

1. *Each module or mode can be associated with a positive-definite Lyapunov function with a negative-definite or negative-semi-definite time derivative.*
2. *All modes of control are associated with the same equilibrium point.*
3. *The Lyapunov function of the mode to be switched in is lower than the Lyapunov function of that mode when it was last switched out.*

The first criterion guarantees that $\mathcal{V}(t+\delta t) \leq \mathcal{V}(t)$ during any specific mode of control. The third criterion guarantees that we never visit any point twice under the same control mode. The second criterion guarantees that there exists a defined equilibrium point.

That a multi-modular system complying to requirement 10.1 is stable in the sense of Lyapunov can be proved by defining a global Lyapunov function which is valid for the entire system. Such a global Lyapunov function could be the sum of all "system-compatible Lyapunov functions". A *system-compatible Lyapunov function* is a Lyapunov function which is identical to the original one while the module is active, and otherwise replaced by a monotonically decreasing function. An example of a system-compatible Lyapunov function is :

$\mathcal{V}'(t) = \mathcal{V}(t)$ if the module is active, $\mathcal{V}'(t) = \mathcal{V}(t_{next-switched-in}) +$

$(\frac{1}{2}(\mathcal{V}(t_{previously-switched-out}) - \mathcal{V}(t_{next-switched-in}))$

$(1 + cos(\pi(\frac{t - t_{previously-switched-out}}{t_{next-switched-in} - t_{previously-switched-out}})))$, otherwise

where $t_{previously-switched-out}$ is time the module previously was active, and

$t_{next-switched-in}$ is the time the module will become active again. It should be noted that for $t > t_{last-switched-out}$, where $t_{last-switched-out}$ is the last time the module will be switched out from, $t_{next-switched-in}$ is undefined. In this case $\mathcal{V}'(t) = \mathcal{V}(t_{previously-switched-out})$.

The corresponding global Lyapunov function would be:

$\mathcal{GV}(t) = \mathcal{V}_j(t) + \sum_{i \neq j}^{N}(\mathcal{V}_i(t_{i,next-switched-in}) +$

$(\frac{1}{2}(\mathcal{V}_i(t_{i,previously-switched-out}) - \mathcal{V}_i(t_{i,next-switched-in}))$

$(1 + cos(\pi(\frac{t - t_{i,previously-switched-out}}{t_{i,next-switched-in} - t_{i,previously-switched-out}}))))$,

where j corresponds to the currently active module, $t_{i,previously-switched-out}$ is time module i previously was active, and $t_{i,next-switched-in}$ the time module i becomes active next. This Lyapunov function is continuous and positive-definite, if all $\mathcal{V}_i(t)$ are continuous and positive-definite, and its time derivative is also continuous and negative-semi-definite if all $\dot{\mathcal{V}}_i(t)$ are continuous and negative-semi-definite and we comply to the rules above. This can be seen if the differentiation is carried out.

Figure 10.7 illustrates three different Lyapunov functions $\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3$ for three different modules. The system consisting of these three modules complies with the given rules. However, the individual modules might increase their respective Lyapunov function while not active, so simply adding the Lyapunov functions together will not necessarily give us a global Lyapunov function which always is decreasing. Figure 10.7 also shows the $\mathcal{V}_1', \mathcal{V}_2', \mathcal{V}_3'$ functions

which are continuous, and monotonically decreasing, and therefore the sum of $\mathcal{V}_1', \mathcal{V}_2', \mathcal{V}_3'$ is a global Lyapunov function.

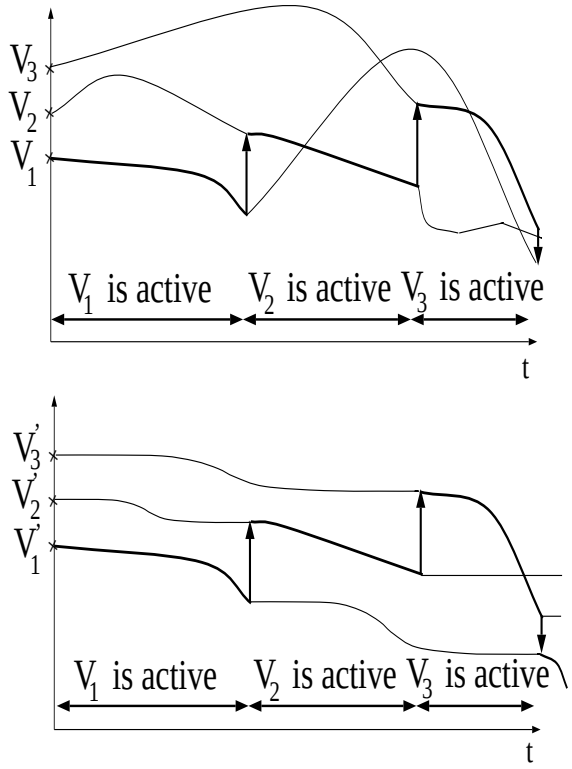


Figure 10.7: Three different Lyapunov functions for three different modules. The system consisting of these modules complies with the requirement above. Adding them together will not give a valid global Lyapunov function. However, adding their system compatible Lyapunov functions $\mathcal{V}_i'$ will give a monotonically decreasing and continuous Lyapunov function.

A more elegant way to show this will be given next. In [10] Michael Branicky proves a theorem which states the stability with respect to an equilibrium point in the case of multiple Lyapunov functions assuming these criterion. His theorem and its proof are given below.

Theorem 10.5 *Suppose we have several Lyapunov functions* V_i, $i = 1, \ldots, N$,

with the same point of global minimum (the origin for convenience), corresponding to the continuous-time vector fields $\dot{x} = f_i(x)$. Let s_k be the switching times of the system. If, whenever we switch in mode (or region) i, with corresponding Lyapunov function V_i, we have $V_i(x(s_k), s_k) \leq V_i(x(s_j), s_j)$, where $s_j < s_k$ is the last time we switched out of mode (or region) i, then the system is stable in the sense of Lyapunov.

Proof Let $R > 0$ be arbitrary. Let $m_i(\alpha)$ denote the minimum value of V_i on $S(\alpha)$. Pick $r_i < R$ such that in $B(r_i)$ we have $V_i < m_i(R)$. This choice is possible via the continuity of V_i. Let $r = \min(r_i)$. With this choice, if we start in $B(r)$, either vector field alone will stay within $B(R)$.

Now, pick $\rho_i < r$ such that in $B(\rho_i)$ we have $V_i < m_i(r)$. Set $\rho = \min(\rho_i)$. Thus, if we start in $B(\rho)$, either vector field alone will stay in $B(r)$. Therefore, whenever the other is first switched on we will have $V_i(x(s_1), s_1) < m_i(R)$, so that we will stay within $B(R)$.

The proof for general N requires N concentric circles constructed as the two were above. □

To illustrate this theorem, suppose we are using the augmented task-space potential function in conjunction with the reflexive filter. Before the augmented task-space guiding potential function would be able to give the control back to the reflexive command filter, $\mathcal{V}_{ref-filt}(\vec{q}_s)(t) < \mathcal{V}_{ref-filt}(\vec{q}_s)(t_{last-switched-out})$ must be true, where $\mathcal{V}_{ref-filt}(\vec{q}_s)(t)$ is the current value of $\mathcal{V}_{ref-filt}(\vec{q}_s)$ and

$\mathcal{V}_{ref-filt}(\vec{q}_s)(t_{last-switched-out})$ the value of $\mathcal{V}_{ref-filt}(\vec{q}_s)$ when we last switched out of the reflexive command mode. If we were switching back to the augmented task-space guiding potential function mode $\mathcal{V}_{guid-pot}(t) < \mathcal{V}_{guid-pot}(t_{last-switched-out})$ must be true instead.

A less restrictive but still useful criteria is:

1. Each module or mode can be associated with a positive-definite, Lyapunov function with a negative-semi-definite time derivative. The derivative must be negative-definite if we also wish to show convergence.

2. The Lyapunov function of the mode to be switched in is lower than the Lyapunov function of that mode when it was last switched out.

In this case it is not required that the Lyapunov functions have the same equilibrium point. We can no longer show stability with respect to a specific point. However, we can still show that all trajectories will converge to at least one of the equilibrium points using the invariant set theorem.

To prove that all trajectories converge to the union of the equilibrium points, using the invariant set theorem, we need to show the following:

1. There exists a function $\mathcal{V}(\vec{x}_s)$ with continuous first partial derivatives.

2. $\mathcal{V}(\vec{x})$ is bounded in the region in question.

3. $\dot{\mathcal{V}}(\vec{x})$ is continuous and negative-semi-definite.

4. $\vec{f}$ is continuous.

The first three criteria are true for the global Lyapunov function given above. It should be noted that the fourth criteria is used to show that $\dot{\mathcal{V}}(\vec{x})$ is uniformly continuous in the proof for the set-invariant theorem. Therefore the fourth criteria can be replaced by the requirement that $\dot{\mathcal{V}}(\vec{x})$ is uniformly continuous, which will be true for the multi-modular system above if all $\dot{\mathcal{V}}_i(\vec{x})$ are uniformly continuous.

10.4 Stability in the Sense of Lagrange and Quasi Lagrange

The concept of stability in the sense of Lagrange, and the local invariant-set theorem are more useful than Lyapunov functions in the context of behavioral, or reflexive systems. The reason is that such systems commonly cannot be associated with a single equilibrium point, but instead sets of attraction. An example of this is the flee reflexes described in Chapter 8.

In Section 10.1 it was described how the local invariant set theorem could be used to show convergence for the flee reflexes. In this section I will also introduce a stability concept which could be used in the context of invariant sets, and sets for which $\dot{\mathcal{V}}(\vec{x}) = \vec{0}$. This type of stability concept will be referred to as Quasi Lagrangian.

The Quasi Lagrangian stability concept is defined with respect to regions, or subsets of the space spanned by higher-level output vectors, rather than equilibrium points. Before this is done we have to define the concept of an "encapsulating superset" and "encapsulated subset." Figure 10.8 illustrates the concept of an encapsulating superset.

Definition 10.16 *Set $\mathcal{A}$ is an* encapsulating superset *of set $\mathcal{B}$ if $\mathcal{A}$ is a superset of $\mathcal{B}$, and for any point $\vec{b}$ belonging to $\mathcal{B}$ there exists a ball B_b with the center $\vec{b}$ and radius $R_b > 0$ $(\forall \vec{x}, \|\vec{x} - \vec{b}\| < R_b)$ such that all points in B_b belongs to $\mathcal{A}$. In other words the boundary of $\mathcal{A}$ does not intersect the boundary of $\mathcal{B}$. In a similar fashion $\mathcal{B}$ is an* encapsulated subset *of $\mathcal{A}$. I will denote this $\mathcal{A} \sqsupset \mathcal{B}$.*

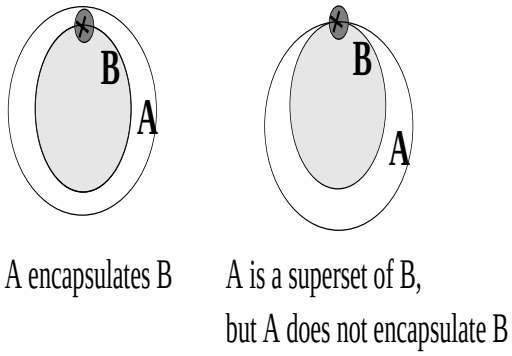


Figure 10.8: In the figure to the left, A is an encapsulating superset of B, because we can find a non-zero sized ball, even for a boundary point of B, which is a subset of A. In the figure to the right this is not possible.

In the following definition I will denote $\mathcal{B}$ is a subset of $\mathcal{A}$ as $\mathcal{B} \subset \mathcal{A}$, and $\mathcal{A}$ is a superset of $\mathcal{B}$ as $\mathcal{A} \supset \mathcal{B}$. That the point $\vec{p}$ is in a set $\mathcal{A}$ will be denoted

$\vec{p} \in \mathcal{A}$. The definition for region stability is as follows:

Definition 10.17 *A system is said to be* stable in a region sense *with respect to the region B_Ω if, for any region B_R being an "encapsulating superset" of B_Ω, there exists a region B_r which is a superset of B_Ω, such that if $\vec{x}(0)$ is in B_r, then $\vec{x}(t)$ is in B_R for all $t \leq 0$, where $\vec{x}$ is a state vector, or higher level output vector.*

or in a more compact form,

$$\forall B_R \sqsupset B_\Omega, \exists B_r \supset B_\Omega : \vec{x}(0) \in B_r \Rightarrow \vec{x}(t) \in B_R, \forall t \geq 0$$

It should be noted that for this definition to be fulfilled it is necessary that $B_R \supset B_r \supset B_\Omega$. Using this definition we can formulate a region stability criterion for using the Quasi Lagrange function. Region stability means that if we start sufficiently close to a region B_Ω we will stay within an arbitrary region B_R encapsulating the region B_Ω. The entire purpose of this definition is to extend the concept of stability to the case where a goal is a subset of the space spanned by the output vectors from higher-level modules, rather than just a single point. The concept of region stability is illustrated in Figure 10.9.

Theorem 10.6 *If, in the ball B_R, there exists a scalar function $\mathcal{Q}$ with continuous first partial derivatives such that,*

1. *$\mathcal{Q}(\vec{x})$ is bounded and continuous everywhere within B_R.*

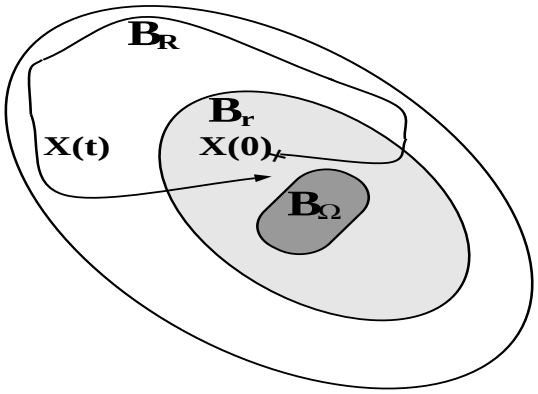


Figure 10.9: The concept of region stability replaces the equilibrium point with a goal set. This goal set could for example be an invariant set. The figure illustrates that for the B_R chosen there is a $\vec{x}(0) \in B_r$ so that $\vec{x}(t) \in B_R$ for all future.

2. *$\mathcal{Q}(\vec{x})$ is strictly positive everywhere inside B_R, except inside B_Ω where it is zero.*

3. *$\dot{\mathcal{Q}}(\vec{x})$ is negative, or zero everywhere.*

then the system described by the Quasi Lagrangian function $\mathcal{Q}(\vec{x})$, is region stable with respect to the region B_Ω. If, actually, the derivative $\dot{\mathcal{Q}}(\vec{x})$ is strictly negative in B_R, then the region stability is asymptotic.

Proof Let m be the minimum of $\mathcal{Q}$ on the boundary of the region B_R. Since $\mathcal{Q}$ is strictly positive outside the region B_R, and B_R is an encapsulating superset of B_Ω, m exists and is strictly positive. Furthermore, since $\mathcal{Q}(B_\Omega) = 0$, there exists a region B_r which is a superset of B_Ω such that $\mathcal{Q}(\vec{x}) < m$ for any $\vec{x}$ inside the region B_r. Since $\mathcal{Q}$ is non-increasing along system trajectories, $\mathcal{Q}$ remains strictly smaller than m, and therefore the trajectory cannot possibly

cross the outside of region B_R. Thus any trajectory inside the region B_r remains inside the region B_R for all future times, and therefore region-stability is guaranteed. □

It should be noted that the Quasi Lagrange functions for the flee reflexes discussed above fulfills the criteria given in this theorem except for two details. The boundary of the region B_Ω is discontinuous, and B_Ω is an unbounded subset of joint-space which cannot be encapsulated by another subset in joint space. This can be taken care of (in theory) by virtually smoothing the flee-potential at the discontinuity, and bounding the region B_Ω so that it covers all space of operation. This will be further discussed in Section 10.4.1.

10.4.1 Stability and Convergence Analysis for the Flee Reflexes

In this section it will be shown that the flee reflexes converges to the invariant set corresponding to the free-space, using the local invariant set theorem. It will also be shown that the flee reflexes are region stable using Quasi Lagrange functions. The stability analysis in this section requires fixed higher-level commands, and a static environment. With a static environment is meant that the obstacle map and the corresponding potential fields for the moving obstacle causing the flee-response are constant.

In the case of the wall-emulating flee reflexes the reflexive command filter

guarantees, under static conditions (the wall not approaching), that the robot will stay within the free space if it already is there. For this reason the free-space corresponds to an invariant set. Further, if the robot is intersecting a wall, the flee reflex will drive the robot into the invariant set (free-space). To show this we can use the flee potential as our function $\mathcal{V}$.

According to Section 8.1 the startle reflex generates a command which brings the robot in a direction which corresponds to a maximum move in the negative y-direction with minimum joint moves. Even though this does not correspond to a move along the y-coordinate axis, it certainly corresponds to a decrease of the y-coordinate, and thus a decrease of $\mathcal{V}$. The retraction reflex moves the robot strictly along the y-axis, which of course corresponds to a decrease of $\mathcal{V}$.

For these reasons, the flee reflex guarantees that the flee potential decreases outside the invariant set, i.e. $\dot{\mathcal{V}} < 0$. Another requirement for the local set-invariant theorem to apply is that $\dot{\vec{x}} = \vec{f}(\vec{x})$ is continuous, which cannot be done.

However, the fact that $\vec{f}(\vec{x})$ is continuous is only used for the purpose of proving that $\dot{\mathcal{V}}$ is uniformly continuous. Thus, the demand that $\dot{\vec{x}} = \vec{f}(\vec{x})$ is continuous can be replaced with the demand that $\dot{\mathcal{V}}$ is uniformly continuous, which in this case is easier to show. When the path generated by the reflex controller is smooth, and continous, and has a bounded time derivative, both

$\mathcal{V}$ and $\dot{\mathcal{V}}$ are uniformly continuous. Thus, the invariant set theorem applies, which means that all trajectories will converge to the invariant set, which in this case is the free-space.

In the case of the block-emulating flee reflex the invariant set consists of the intersection of the free-space and the space where the flee potential is zero. The static command filter guarantees that the robot always will stay within the free-space once in the free-space. The hold reflex guarantees that the robot will not approach the flee potential field once the robot has left the flee potential. The hold reflex and the static command filter are the two "containment reflexes"[5] which define the invariant set in this case.

It should be noted, however, that the intersection of the free-space and the space where the flee potential is inactive does not correspond to a true invariant set, due to the fact that robot dynamics can cause the robot to enter the flee potential, even though it cannot reenter once it left the flee potential. However, assuming that the robot has entered the flee potential once, the union of the free-space and the space where the flee potential is inactive represents an invariant set.

The block-emulating flee reflex also consist of a startle reflex, and a retraction reflex. Both the startle reflex and the retraction reflex guarantee that $\dot{\mathcal{V}} < 0$, for the same reason as for the wall-emulating flee reflex. In other

[5]For a definition of containment reflexes see Section 10.5, or 3.3

words, both the wall-emulating flee reflex, and the block-emulating flee reflex guarantee that all trajectories will converge to an invariant set, according to the local invariant set theorem.

In the remainder of this section, we will discuss the region stability of the flee reflexes. To show region stability, of the flee reflexes, we need to show that for the corresponding Quasi Lagrange functions the following criteria holds for the retraction reflex.

1. $\mathcal{Q}(\vec{x})$ is bounded, continuous, and has continuous first partial derivatives, everywhere within B_R.

2. $\mathcal{Q}(\vec{x})$ is strictly positive everywhere inside B_R except inside B_Ω where it is zero.

3. $\dot{\mathcal{Q}}(\vec{x})$ is negative, or zero everywhere.

according to Theorem 10.6.

The startle reflex is a reflex which is fired once and then turned off and replaced with the retraction reflex. For this reason the startle reflex can be ignored in this stability analysis. The Quasi Lagrange function for the wall-emulating retraction reflex is $\mathcal{Q}_{retr}(\vec{x}) = (y_{field-off} - y(\vec{x}))$ if $y(\vec{x}) \leq y_{field-off}$. The connected goal subset is in this case, $y(\vec{x}) \leq y_{field-off}$. The analysis in Section 10.4 referred to goal subsets limited in size which could be enclosed by regions were the Quasi Lagrange function is defined. This is obviously not the

case here. However, this is still not a problem. For the purpose of analysis we can limit the goal subset, and surround it with the Quasi Lagrange function, in a manner which has no practical implications[6].

$\mathcal{Q}_{retr}(\vec{x})$ is obviously continuous and positive everywhere except at the goal set, where $\mathcal{Q}_{retr}(\vec{x}) = 0$. Further, the retraction reflex commands a motion of the robot's tool which is parallel to the y-axis, and thus $y(\vec{x})$ is monotonically increasing, which means that $\dot{\mathcal{Q}}_{retr}(\vec{x})$ is negative, and thus the system is asymptotically stable in a Quasi Lyapunov sense.

In the case of the block-emulating flee reflex the Quasi Lyapunov function is $\mathcal{Q}_{retr}(\vec{x}) = \mathcal{F}_{flee}(\vec{x})$ when $\vec{x}$ is located within the reach of the flee-potential field and otherwise 0. $\mathcal{Q}_{retr}(\vec{x})$ is obviously continuous and positive except for outside the flee-potential. When the retraction reflex in this case commands a motion which is derived from the gradient of the flee-potential, $\dot{\mathcal{Q}}_{retr}(\vec{x})$ is Quasi negative-definite, and thus the system is asymptotically stable in a Quasi Lyapunov sense.

[6]For example limiting the goal subset so that the space belonging to $y(\vec{x}) \leq y_{field-off}$ outside the goal subset could never be reached by the robot

10.5 Stable Interaction Among Simple and Triggered Containment Reflexes, and Simple or Triggered Repulsors

In this section I will discuss stability and cycling with respect to interaction among simple-containment reflexes and triggered-containment reflexes, and repulsor reflexes. This analysis is not applied in real time, and is therefore fundamentally different from, for example, Lyapunov theory. Only the order in which things happen matters.

The definitions for simple, triggered, containment, and repulsor reflexes are:

- Simple reflexes are defined as reflexes which are active in a specific region of the state-space called the activation region of the reflex. It should be noted that the activation region might change due to external stimuli, or a changing environment. However, the activation region is not dependent on the current robot state, or internal task space commands.

- Triggered reflexes are reflexes which are activated if the robot enters a certain region of the state-space called the trigger region, and remains active as long as the robot remains in a certain region of the state-space called the activation region. The activation region must be a superset of the trigger region. In other words the activation region has hysteresis.

The difference between the activation region and trigger region is called the hysteresis region. It should be noted that if the trigger region and the activation region are identical the reflex is a simple reflex. This type of reflex is illustrated in Figure 10.10.

- A containment reflex is a reflex for which all commands generated by the reflex are confined to the same region in which the reflex is active. A containment reflex can, for example, be a simple or a triggered reflex. Examples of simple-containment reflexes are:

  1. Reflexes which generate commands which bring the robot to a specific point.

  2. Reflexes which make the robot converge to a subset, rather than a specific point.

  3. A reflex which keeps the robot in the position within the activation region where the robot entered the activation region (first was found within the activation region). This particular type of reflex is called a "hold reflex". The point where the reflex "holds" the robot does not have to be located on the boundary of the activation region. External stimuli can, for example, create a new activation region. Sampling times and delays can also allow the robot to penetrate a fixed activation region.

4. Reflexes which confine the robot to the activation region, like the static command filter (general case).

Figure 10.11 illustrates these four simple-containment reflexes.

- Repulsor reflexes push the robot outside the activation region. A repulsor reflex can be a simple or a triggered reflex.

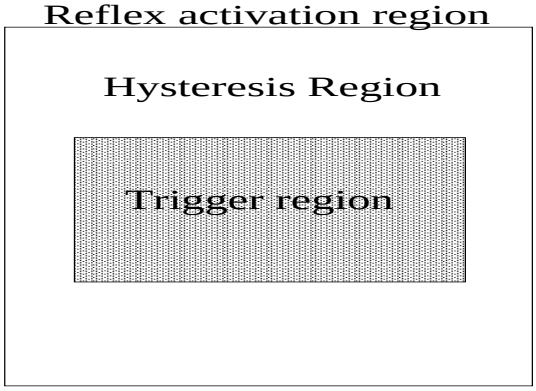


Figure 10.10: If the robot enters the grey area the reflex is activated. This is called the trigger region. The reflex is not deactivated until the robot leaves the white area. The white area is called the hysteresis region. The activation region is the union of the hysteresis region and the trigger region.

If in a set of interacting reflexes, the reflex R_A controls the robot, the reflex R_A is said to be active. If R_A is a simple reflex, R_A can only be active if the robot is within its activation region A. If the robot is located inside the intersection of two activation regions, the reflex which controls the robot is said to have precedence over the other.

Assume three simple-containment reflexes R_A, R_B, and R_C. Assume reflex

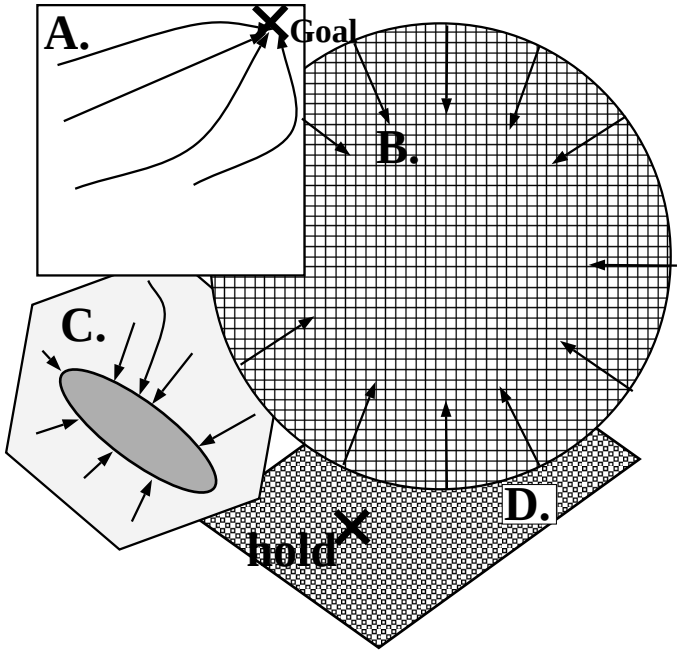


Figure 10.11: Four reflexes R_A, R_B, R_C, and R_D are active in four different regions. R_A generates a specific command, R_B keeps the robot within the activation region, R_C makes the robot converge to a subset of the activation region, and R_D holds the robot where it entered the activation region.

R_A has precedence over reflex R_B when the regions for reflex R_A and R_B overlap, and reflex R_B has precedence over reflex R_C when their corresponding regions overlap. If reflex R_C has precedence over reflex R_A when their respective regions overlap, this could lead to cycling. This situation is illustrated in Figure 10.12. However, if reflex R_A has precedence over R_B and R_C, and R_B has precedence over reflex R_C, then the reflexes will interact without causing cycling. This result can be generalized in the theorem below. However, I first need to make a definition of set connectedness and reflex precedence.

Definition 10.18 *Two sets A and B intersect if $A \cap B \neq \emptyset$. If there exists*

a set of sets S_i such that S_0 intersects S_1, and S_i intersects S_{i+1}, and S_{N-1} intersects S_N, then S_0 and S_N are connected.

An alternative way to define connectedness is through the use of a recursive definition.

Definition 10.19 *Set A and set B are connected if they intersect, or A intersect with a set which is connected to B.*

It should be noted that if a set A is connected to a set B, and B is connected to a set C, then set A and C are connected. This definition is illustrated in Figure 10.13.

Definition 10.20 *Reflex R_A precedes reflex R_B if R_A generates the system command in areas where the two reflexes' activation regions intersect. If the two reflexes do not have any intersecting activation region, the precedence can be randomly set. If reflex R_A precedes reflex R_B we denote this $R_A > R_B$.*

Theorem 10.7 *Suppose a system consists of a set of simple-containment reflexes R_i with connected activation regions in the state-space. If all reflexes R_i can be arranged in a precedence order $R_j > R_i$, if $j > i$, then the connected set of reflexes R_i does not contain any cycling among its activation regions.*

It should be noted that if two reflexes R_k and R_j do not intersect, the precedence order among them can be chosen arbitrarily. However, if $R_k > R_m$, and

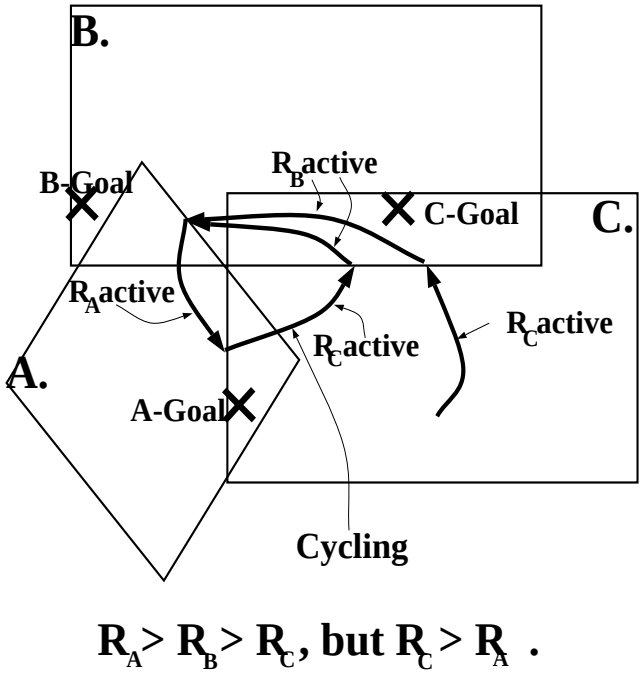


Figure 10.12: If $R_A > R_B$, $R_B > R_C$, and $R_C > R_A$, this could lead to cycling.

$R_j < R_m$, we have to choose $R_k > R_j$. The list containing the precedence of all the reflexes will be denoted the *precedence list* in this text.

Proof The following is true by definition if the reflexes are ordered in a precedence order $\cdots > R_{i+1} > R_i > R_{i-1} > \cdots$:

Suppose the robot is controlled by the reflex R_i. The reflex R_i generates commands which are constrained to its activation region. Thus the reflex R_i cannot loose control of the robot unless a reflex R_j of higher precedence takes control of the robot. Thus the precedence level of the reflex in control can never decrease. This means that none of the reflexes can be reactivated, which means that we cannot have cycling with respect to reflex activations. □

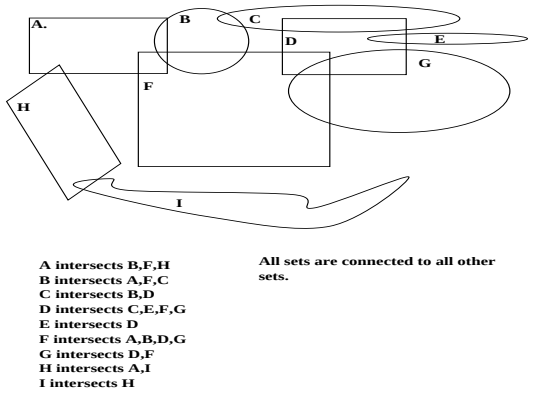


Figure 10.13: A connected set of regions.

Trigger-containment reflexes are defined as reflexes in which activation regions depend on whether the reflex already is active or not. In other words the activation region has hysteresis. If the reflex is not active, the reflex will become active as soon as the robot reaches a certain region called the *trigger region*. If the robot is already active, it will remain active as long as it resides within the region called the *activation region*. The trigger region must be a subset of the activation region. This is illustrated in Figure 10.10. The difference between the trigger region and activation region is called the hysteresis region.

Trigger-containment reflexes can be treated the same way as simple-containment reflexes with respect to precedence lists. The reason is that, once a trigger-containment reflex has been triggered (activated), only reflexes with higher precedence can steal the control of the robot. It is thus impossible to retrigger the trigger-containment reflex again.

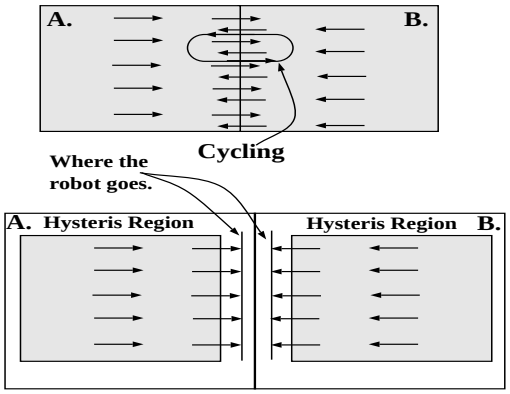


Figure 10.14: If the purpose of two adjacent reflexes R_A, and R_B is to bring the robot outside the activation regions, instability can result, with or without a precedence list. The precedence list does not matter in any of the cases when the activation regions do not intersect.

For the case of repulsor reflexes, a precedence list will not be enough to guarantee stability. The reason is that the repulsor reflex will move the robot outside its activation region. When the robot is outside the activation region, other reflexes might be activated, which will drive the robot back into the activation region of the repulsor reflex. However, stability is achieved if all reflexes with activation regions intersecting the activation region of the repulsor reflex, are made compatible with the repulsor region.

Simple-containment reflexes can be made compatible with a repulsor reflex if one of the following is true:

- The repulsor reflex has a lower precedence than all of the simple-containment reflexes which it intersects. In other words, once the repulsor reflex is done, it cannot take control of the robot again.

- The boundary of the activation region of the repulsor reflex is compatible

with all other reflexes intersecting its boundary. In other words, the commands generated by adjacent reflexes will not drive the robot back into the activation region of the repulsor reflex. A hold reflex is always compatible with all types of repulsor reflexes. If the adjacent reflexes are not hold reflexes, the trajectories generated by the adjacent reflexes must correspond to a positive or zero, dot product with the boundary normal of the repulsor reflex.

The system consisting of the block-emulating flee reflex, the hold reflex, and the static command filter can be analyzed using the techniques developed in this section.

- The block-emulating flee reflex can be viewed as a triggered repulsor reflex, which drives the robot outside its activation region as soon as the robot enters its trigger region.

- The hold reflex can be viewed as a simple-containment reflex. The activation region for the hold reflex is a superset of the activation region for the block-emulating flee reflex. The block-emulating flee reflex has precedence over the hold reflex. However, a hold reflex is compatible with any repulsor reflex.

- The unspecified higher-level module[7] for which it generates robot commands in joint space, can be viewed as a simple-containment reflex active

[7]For example a human operator

everywhere. It has the lowest precedence of all reflexes.

- The reflexive command filter can be viewed as a simple-containment reflex which activation region is the free-space. However, the reflexive command filter is not a reflexive action competing with the other reflexes. It is a command filter located one step below the other reflexes in the system hierarchy, as indicated in Figure 10.15. It has precedence over all other reflexes due to this fact. However, it cannot be included in the precedence list analysis with the other reflexes, when its precedence over the other reflexes does not deactivate these reflexes.

The activation regions for the different reflexes are illustrated in Figure 10.15.

This system is stable because the two containment reflexes, the hold reflex, and the command module, have a specific precedence order, and the hold reflex is compatible with the repulsor reflex (the flee reflex). The static command filter resides at a lower level in the system hierarchy.

10.6 The Use of Progress Measurement Functions

The progress measurement function is another concept which is used for the purpose of non-real time control analysis of interacting modules. It is a more general concept than the concept described in Section 10.5.

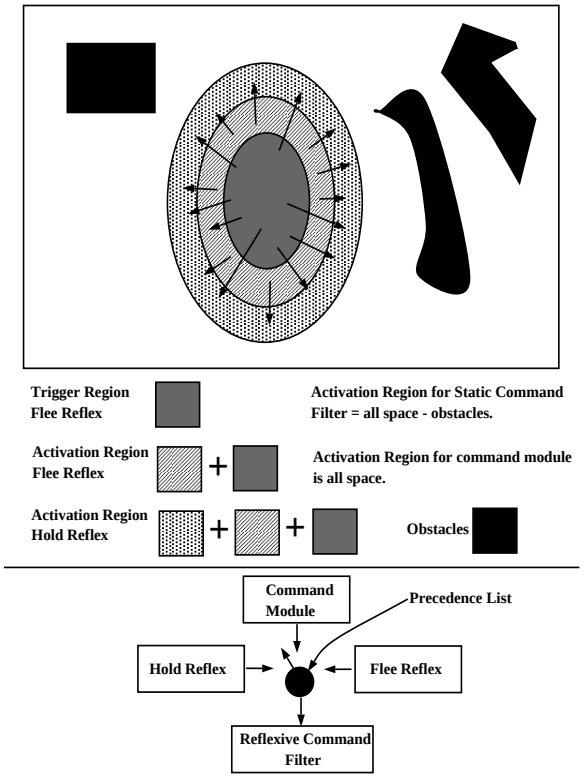


Figure 10.15: The respective activation regions for the flee reflex, hold reflex, command module, and reflexive command filter. The reflexive command filter resides at a lower level in the system hierarchy.

The *progress measurement function* denoted $\mathcal{P} = \mathcal{D} + \mathcal{RT}$ consists of two parts.

- The distance measurement $\mathcal{D}$ measures the distance to a certain system goal. This can be the actual distance to a goal in state-space, or a measurement of how much more must be done to complete a certain task. A natural choice for $\mathcal{D}$ for the example of a robotic lawn mower would be the lawn area which has not yet been mowed.

- The readiness for task measurement $\mathcal{RT}$ measures how difficult it is decrease the distance measurement $\mathcal{D}$. The reason for the existence of $\mathcal{RT}$ is that the purpose of many modules in complex systems is not to complete the system task, but instead to enable other modules to do so.

The progress measurement function is used to make sure that the activation of a module will, at the time it is done, result in a decrease of the progress measurement function. While the module is active it is allowed to temporarily increase the progress measurement function. Only the value of the progress measurement function at control mode switches is of interest. It should be noted that the progress measurement function does not correspond to a control method; it is merely an analytical tool used to show that a given system will converge to a system goal. It should also be noted that the concept of the progress measurement function applies to any kind of module, fixed action patterns, and other behavioral modules.

A simple illustration of a progress measurement function is the following. Suppose a robot (artificial insect, mobile robot etc.) must reach a certain final position. The control system consists of two modules. One module A, is able to generate a pure rotation around a z-axis of the robot. The other module B, is able to move the robot forward (only). This situation is illustrated in Figure 10.16. If the robot is turned the wrong way module A must first rotate the robot.

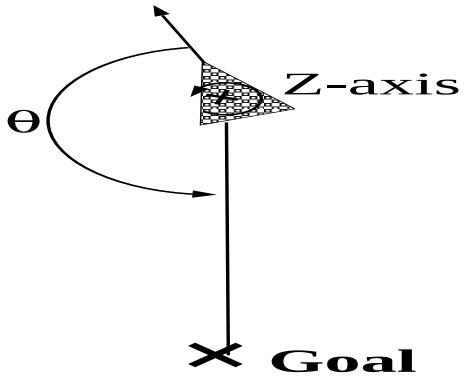


Figure 10.16: The robot in the picture can only move forward and therefore must be turned towards the goal before moving.

The progress measurement function in this case could for example be $\mathcal{P} = \|(\vec{x}_0 - \vec{x}_g)\| + \|(\vec{x}_0 - \vec{x}_g)\|\|sin(\frac{1}{2}\theta)\|$, where $\vec{x}_0$ is the start position, $\vec{x}_g$ is the final position, and θ the angle between the orientation of the robot and the direction towards the goal. Until $\theta < 45$ degrees module B cannot decrease the progress measurement function, and should therefore not be activated. However, module A can decrease the RT component of the progress measurement function, and can therefore be activated. If $\theta = 0$ degrees, module A can no longer decrease the progress measurement function, but module B can do so.

Progress measurement functions are not necessarily expressed in terms of state-space vectors, or as desired state vectors. For this reason it is important to make sure that a decrease of the progress measurement function corresponds to what we would like to mean by progress. For example $\frac{1}{N}$, where N corresponds to the number of times we are switching control modes, is not a permissible component of a progress measurement function.

Progress measurement functions are generated as follows:

1. Construct a distance measurement D which corresponds to how much is left to do before a task is completed. Make sure that D cannot be decreased by purposeless mode switching. A decrease of D must correspond to bringing the task closer to completion. The distance measurement must be zero at the goal.

2. Construct a measurement which corresponds to how difficult it is to decrease the distance measurement. This measurement must also be zero at the goal.

3. Add the two.

4. Make sure all modules are able to decrease the progress measurement function, and that at least one of the modules is able to decrease the progress measurement function in all possible situations.

When analyzing a control strategy one must make sure that:

1. The requirements for the progress measurement function are met.

2. A module is only activated if it can be proven that the activation of the module finally will lead to a decrease of the progress measurement function.

3. The decrease of the progress measurement function is larger than a chosen minimum value $\Delta\mathcal{P}$.

If this is true, the system will converge to the system goal in a finite time less than $\frac{\mathcal{P}_{start} t_{max}}{\Delta\mathcal{P}}$, where $\mathcal{P}_{start}$ is the initial value of the progress measurement function, t_{max} an upper limit on the time any module is active, and $\Delta\mathcal{P}$ the minimum decrease of the progress measurement function.

An example of a progress measurement function is the one for the sonar-based world mapping system,

$$\begin{aligned} \mathcal{P}_0 &= V^U + \sum_i V_i^O (1 - \frac{C_i}{T_0}) \frac{1}{2}(1 + D_i) + (256 - T_0) V^T, \\ \mathcal{P}_n &= (256 - T) \sum_i V_i^O (1 - \frac{C_i}{T}) \frac{1}{2}(1 + D_i), \end{aligned}$$

where $\mathcal{P}_0$ corresponds to the progress measurement function before all space has been explored at least once, and $\mathcal{P}_n$ corresponds to the rest of the time ($\mathcal{P}_n < \mathcal{P}_0$).

V^U is the volume of all unexplored space, V_i^O the volume of the found obstacle i, C_i the confidence level for obstacle i, T_0 the initial confidence threshold level, T the confidence threshold level, and V^T the volume of all space. D_i is a measurement which measures the robot's ability to investigate a found inconsistency. The maximum value of D_i is 1 and the minimum 0. The less D_i is, the better the robot can investigate the inconsistency. D_i can also be seen as a distance measurement to the obstacle. It should be noted that 256 is the

maximum confidence threshold level (the goal). This progress measurement function is zero at the goal (all space fully examined), and any decrease of any of the components corresponds to bringing the the task closer to completion.

When the robot's work-space is partially unexplored, $\mathcal{P} = \mathcal{P}_0$ and the respective module does the following to the progress measurement function:

1. Look-Around will decrease V^U. If an inconsistency is found $\sum_i V_i^O$ is increased with the same amount that V^U is increased. When $(1-\frac{C_i}{T_0})\frac{1}{2}(1+D_i)$ always is less or equal to one this means either a decrease of $\mathcal{P}_0$, or that $\Delta\mathcal{P}_0 = 0$. However, when unexplored space always must be explored before an unknown obstacle can be found, this always means a decrease of $\mathcal{P}_0$.

2. Look-Path will either leave $\mathcal{P}_0$ intact or decrease $\mathcal{P}_0$ depending on whether the desired path goes through already explored or unexplored space. Look-Path must be activated a limited amount of times for the system to be able to converge.

3. Beam-At directs the beam towards a found obstacle and will therefore decrease D_i. It is also likely to decrease V^U.

4. Approach will decrease D_i, and possibly V^U.

5. Investigate will increase the confidence level for the found inconsistency and will therefore decrease the factor $(1 - \frac{C_i}{T_0})$.

Once all space has been examined at least once, $\mathcal{P} = \mathcal{P}_n$. Under this condition the respective module does the following.

1. Look-Around will no longer be active.

2. Look-Path will be active, but not for the purpose of exploring unexplored space. It will not effect $\mathcal{P}_n$. It must therefore only be activated a limited amount of times.

3. Beam-At directs the beam towards the found obstacles and will therefore decrease D_i.

4. Approach will decrease D_i.

5. Investigate will increase the confidence level for the found inconsistency and will therefore decrease the factor $(1 - \frac{C_i}{T_0})$, and ultimately increase T.

To show that $\mathcal{P}$ finally will become zero, we must be able to show that the total operating time which leaves $\mathcal{P}$ intact is finite. If the sonar-based world mapping system continuously receives tasks to complete, which results in the robot always passing through investigated space, this cannot be done.

10.7 Final Conclusions and Overview for Chapter 10

This chapter dealt with various concepts regarding stability, convergence, cycling, and performance in the context of reflexive or behavioral modules. Common stability concepts which apply to real-time single-module systems were reviewed and applied to the reflexive systems I have implemented. New performance measurements were introduced in Sections 10.5 and 10.6. These performance measurements were used to analyze non-real-time interaction among behavioral modules.

The concept of stability in the sense of Lyapunov was discussed and implemented on the static command filter and the guiding potential functions. The interaction between the static command filter and the guiding potential functions was analyzed using global Lyapunov functions, and multiple Lyapunov functions. The invariant set theorem was succesfully used to show that the wall-emulating flee reflexes, and the block-emulating flee reflexes resulted in the robot converging to free-space. The concept of Quasi Lagrange stability was also introduced so that the flee reflexes could be analyzed in terms of stability. Stability and convergence analysis applied to the output vectors of higher-level modules was validated using the concept of command-tolerant stability.

In Section 10.5, the interactions between simple-containment, and triggered-containment and repulsor reflexes were discussed. The system consisting of the flee reflex, hold reflex, command generator, and static command filter was discussed. In Section 10.6, progress measurement functions were used to analyze the performance of the sonar-based world mapping system.

The concepts discussed and introduced in this chapter were applied to the reflexes and behavioral modules I constructed. It was demonstrated that Lyapunov functions, multiple Lyapunov functions, the set-invariant theorem, and (Quasi) Lagrange stability concepts, are useful concepts for real time analysis of behavioral systems. The new concepts introduced were shown to be useful and fairly general.

Chapter 11

Advice and Experience in the Context of the Design of Practical Behavioral Systems

In this chapter I will discuss the stability and cycling problems I encountered during my experiments. I will also give qualitative advice on how to construct reflex modules, and how to design systems containing interacting behavioral modules. First I will give the arbitrary behavioral structure mentioned in Section 3.3 which I will refer to in specific examples later on.

Figure 11.1 illustrates a generic behavioral structure. In this hierarchical structure the behavioral command filters constitute the "stem" through which all commands are passed and filtered. The behavioral action modules are organized in layers. Within each layer the different modules compete for control. These modules may depend on external sensors, and higher level commands, as well as the robot state. These potential dependencies are not indicated in Figure 11.1. The "virtual switch controller" in Figure 11.1 is not necessarily a separate switch controller. In the systems I implemented, the virtual switch controller is simply the result of the interaction among the virtual sensors of

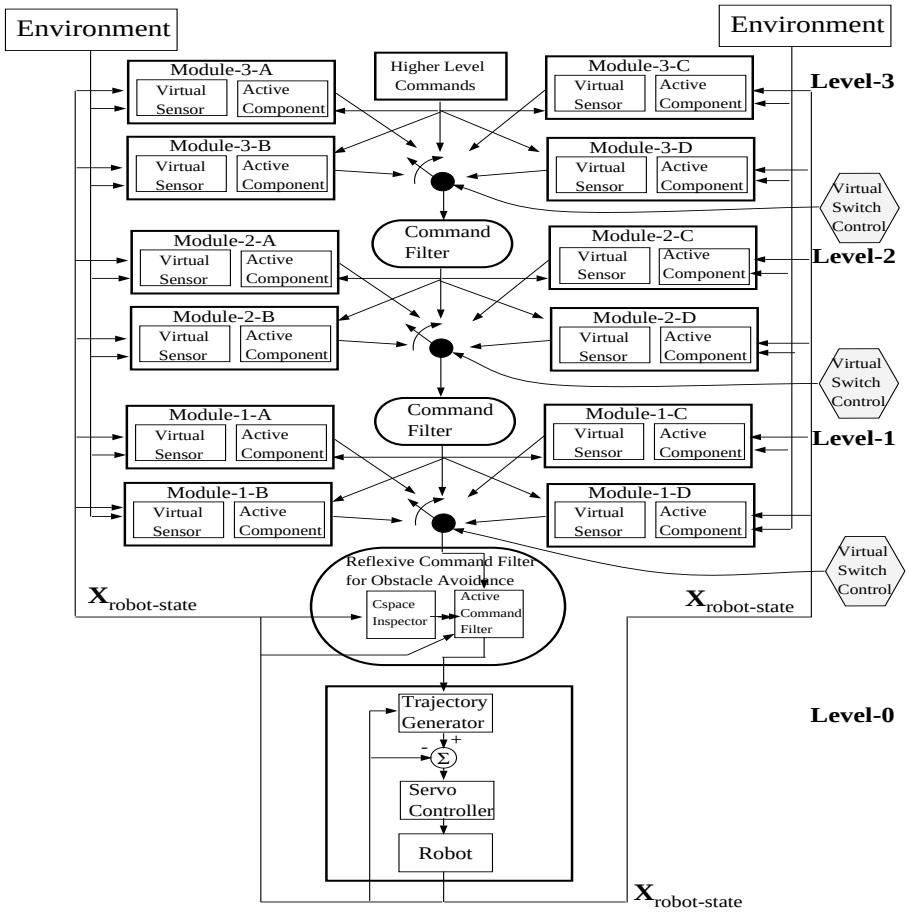


Figure 11.1: An illustration of a generic behavioral structure. The behavioral command filters constitute the "stem" through which all commands are passed and filtered. The behavioral action modules are organized in layers. Within each layer the different modules compete for control. The virtual switch controller represents the interaction among the modules.

the behavioral modules. In other words the individual modules take control of the "virtual switch controller", depending on the state of the virtual sensors. However, it is conceivable that the virtual switch controller represents a separate module, or a human operator.

An example of such a structure is the following. At level-0 resides the servo controller, the robot and the reflexive command filter for collision avoidance.

At level-1 resides the block-emulating flee reflex set, the startle reflex, the retraction reflex, and the hold reflex. A path-planner corresponds to the command filter at this level. The path-planner translates higher-level commands into a set of sub-goals which are attainable using the lower level reflexes. At level-2 resides the sonar-based world mapping system or a human operator.

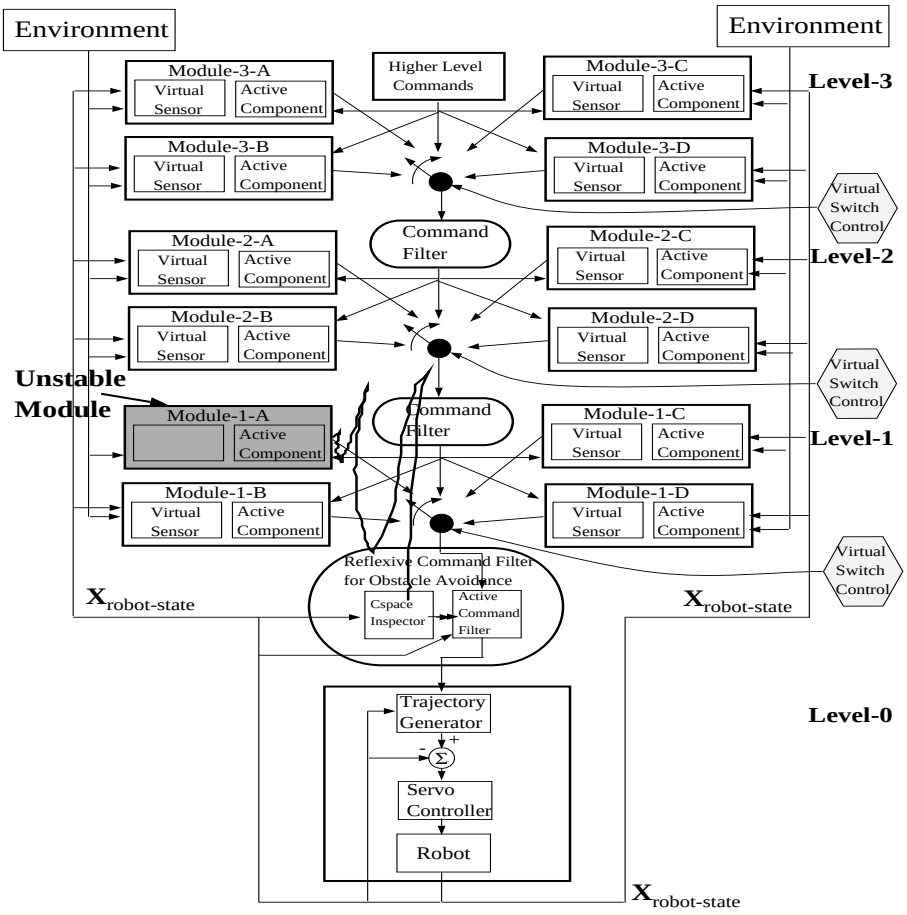


Figure 11.2: A module is badly designed and generates an unstable, and finally unbounded output to the rest of the system, disregarding all feedback loops.

The stability and performance problems I encountered during my experiments were of the following types:

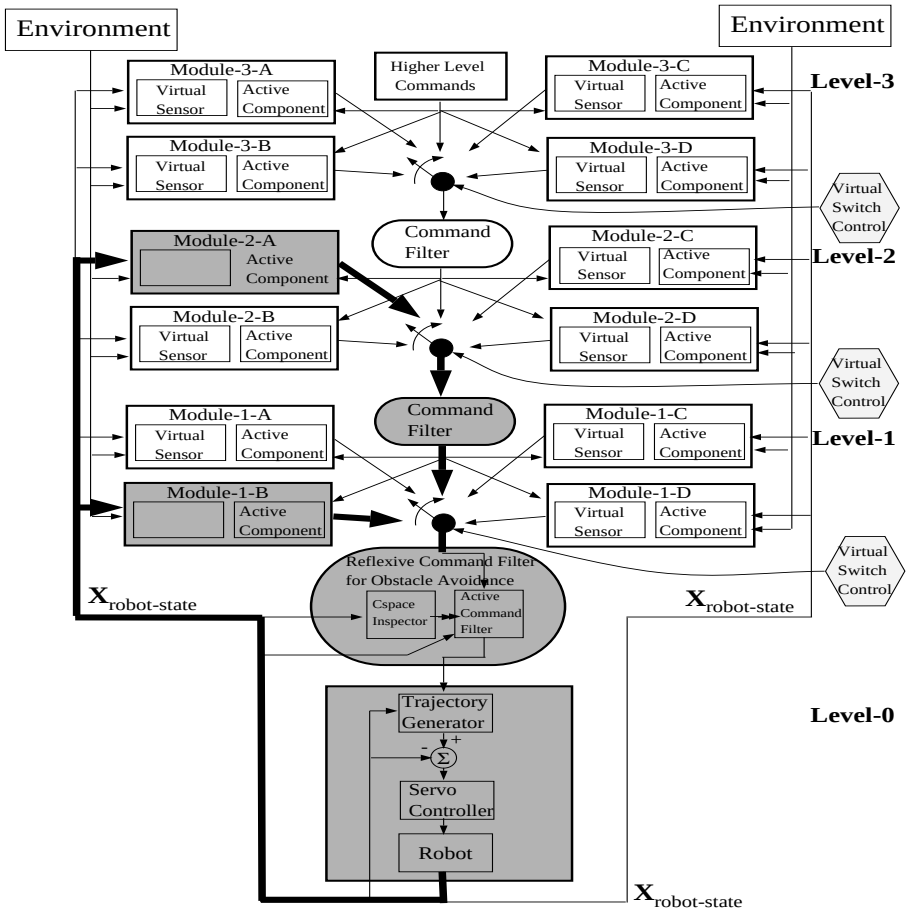


Figure 11.3: The system is unstable due to the unsuccessful closing of two feedback loops.

1. One module, or a set of interacting modules, is badly designed and therefore behaves unstable in open loop. An illustratation of the locality of these types of problems in the pine-tree structure is made in Figure 11.2.

2. One or more modules are unstable in their respective feedback loop. This is illustrated in Figure 11.3.

3. Real-time interaction problems among the modules. This is illustrated in Figure 11.4.

4. Non-real-time interaction problems among the modules. This could also be illustrated by Figure 11.4.

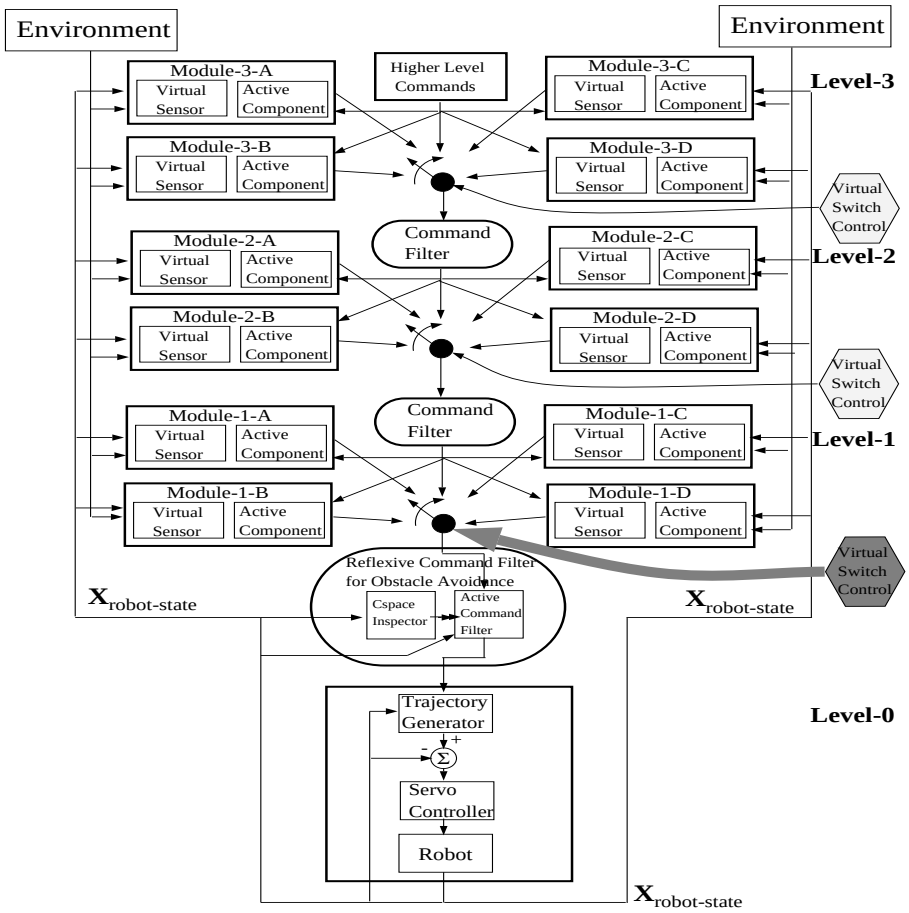


Figure 11.4: The switching among the modules is performed unsuccessfully, and generates cycling. For example, module A is repeatedly reactivated under identical conditions.

Section 11.1, 11.2, 11.3, and 11.4 are devoted to the discussion of respective cases. In Section 11.5 I will give advice regarding the design of reflexive and behavioral systems.

11.1 Single Module Instability

Even though it is common in "low-level" system design to allow a single module to be individually unstable, as long as the resulting system with feedback loops is stable, this is not the case for complex behavioral modular design. The reason is that the individual behavioral modules perform specific tasks which can be functionally isolated, and even used in other systems. In other words behavioral modules are expected to be stable when isolated, as well as in feedback loops. The reasons why individual behavioral modules are unstable, are for example, that the basic module design is flawed, or that the computer program implementing the module contains bugs. A few examples of problems I encountered are :

- In an early development stage the C-space inspector in the reflexive command filter sometimes could not decide on what free-space prism to approve at obstacle corner points. The cause behind this instability was that the C-space inspector, at that time, was allowed to disapprove free-space fronts which already had been approved. The problem was made worse due to the fact that a program bug resulted in disapproval of free-space containing parts of the braking prism. This situation is illustrated in Figure 11.5. This situation could have been resolved at the design stage with the help of Lyapunov analysis.

- Several C-programming bugs during development causing single module instabilities, resulting in system instabilities

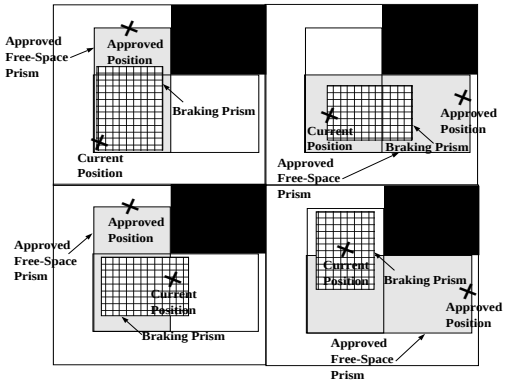


Figure 11.5: The C-space Inspector is undecided about the free-space prisms it generates in this example of a flawed design.

Chapter 10 describes several methods for analyzing these situations. The stability of a single module can be analyzed using Lyapunov functions, the local set invariant theorem, or Quasi Lagrange functions.

11.2 Instability and Cycling Due to Unsuccessfully Closed Feedback Loops

This type of problem can be subdivided into three sub classes:

1. Cycling and stability problems may arise due to the active component's dependency on the robot state. In other words the system consisting of the servo-controller, robot, and the active component of the reflex, with a

feedback loop from the robot state to the active component of the reflex is unstable. This situation is illustrated in the top part of Figure 11.6.

2. Cycling and stability problems may arise due to the virtual sensor's dependency on the robot state. In other words the system consisting of the servo-controller, robot, and the virtual sensor component of the reflex, with a feedback loop from the robot state to the virtual sensor is unstable. This situation is illustrated in the bottom part of Figure 11.6.

3. Cycling and stability problems may arise due to multiple feedback loops.

From classical control theory we know that one way to achieve a stable system containing multiple feedback loops, is to make the inner feedback loops the fastest. In other words, for nested feedback loops the bandwidth for the feedback loop system should be larger the more deeply nested it is. A rule of thumb for linear systems is that the difference in bandwidth between successive loop closures should be an order of a magnitude. This situation is illustrated in Figure 11.7.

The reason behind this is that the innermost feedback loop will appear to be a constant (the trivial plant) to the next outer loop in this setup. The innermost feedback loop system will already have converged before the outer feedback loop starts to settle. Another way to see it, is that it is not possible to derive the velocity from the acceleration before we know anything about

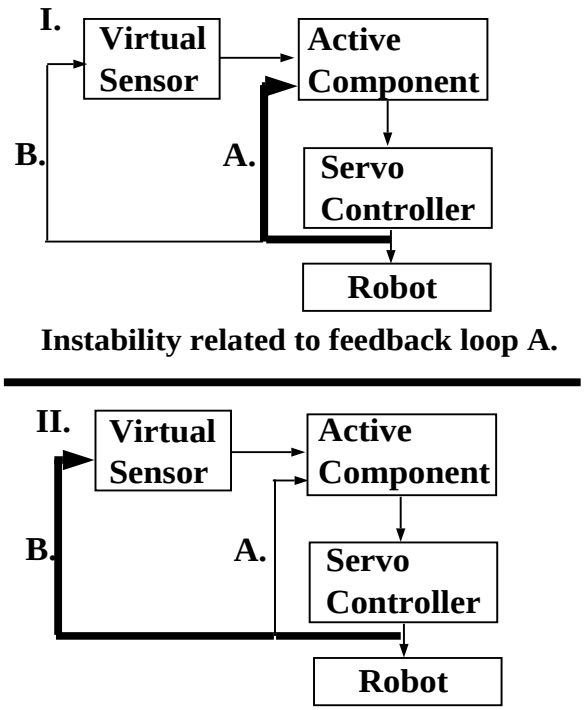


Figure 11.6: Illustration of two configurations that could cause instability : I. Time delays, robot dynamics etc. could cause cycling problems or instability when the active component in the reflex controller depends on the robot state. II. Time delays, robot dynamics etc. could cause instability or cycling problems when the virtual sensor in the reflex controller depends on the robot state.

the acceleration, but if the acceleration is known we can derive the velocity. Something similar can be said about the active component, and the virtual sensor feedback loops.

With respect to Figure 11.7 the transfer functions of each loop closure are:

$$
\begin{aligned}
T_1(s) &= \frac{H(s)}{1 + K_a H(s)} \\
T_2(s) &= \frac{H(s)}{(1 + K_a H(s))s + K_v H(s)}
\end{aligned}
$$

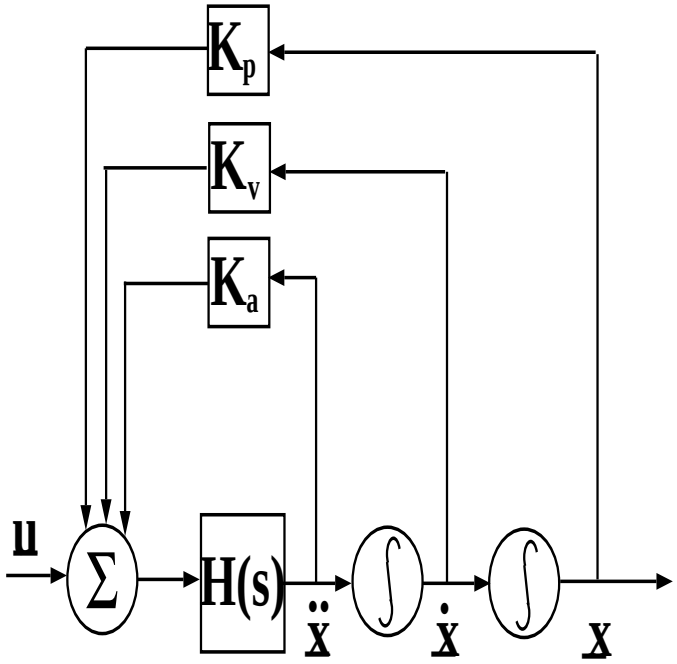


Figure 11.7: Three feedback loops with increasingly smaller bandwidths the further out the loop is.

$$T_3(s) = \frac{H(s)}{((1 + K_a H(s))s + K_v H(s))s + K_p H(s)},$$

where $T_3(s)$ corresponds to the transfer function of the entire system.

It is possible to derive approximate relationships among the constants K_a, K_v, and K_p. If K_a is large enough[1] it will dominate $T_1(s)$ so that $T_1(s)$ corresponds to the trivial plant $\frac{1}{K_a}$. For example, if $H(s) = \frac{1}{s}$ then $T_1(s) = \frac{1}{s+K_a}$, which is a low-pass filter with a bandwidth of K_a. If $K_a \gg \|s\|$ this low-pass filter will act as the trivial plant $\frac{1}{K_a}$. Under these circumstances $T_2(s)$ will act as a low-pass filter with a bandwidth around $\frac{K_v}{K_a}$. For this low pass filter to have a smaller bandwidth than the innermost loop $K_a \gg \frac{K_v}{K_a}$, or $K_a \gg \sqrt{K_v}$.

[1] In the frequency range of interest

For this loop to act as the trivial plant to the next outer-loop, or in other words, act as a low-pass filter with much higher band-width than the outer-loop, $\frac{K_v}{K_a} \gg \frac{K_a K_p}{K_v}$, or $\frac{K_v}{K_a} \gg \sqrt{K_p}$. These relationships are not very meaningful in our context. However, it is important to design the feedback loops so that the fastest loops are the innermost loops.

The active component typically monitors the robot state for the purpose of continuous command generation. For example, the active component of the reflexive command filter recomputes approved commands at a rate of 2000 Hz based on the robot state, higher-level commands, and the approved free-space prism. The virtual sensor feedback loops, on the other hand, generate low bandwidth data, like reflex on-or-off, potential function updates, etc., based on, for example, what region in the robot state space the robot is located. In this analysis this fact will be used to analyze the two types of feedback loops separately.

However, it should be noted that the system consisting of both feedback loops (active component and virtual sensor), can often be analyzed using, for example, Lyapunov functions or Quasi Lagrange functions. This is done in Section 10.3.1 where the stability for the reflexive command filter was shown using a Lyapunov function.

11.2.1 Instability and Cycling in the Active Component Feedback Loop

Cycling and instability problems generated in the feedback loop from the robot output to the active component are very typical for systems containing reflexive actions. The problems are due to unfavorable robot dynamics and time delays in the reflex control loop.

In this case it is usually time delays and robot dynamics which cause instability in the feedback loop. This situation may arise, for example, when the robot is located between two obstacles which both generate a flee-potential and a resulting flee-response. The two flee-potentials will generate a local minimum. Time delays and sampling times would generate a non-zero minimum step size in joint space. With a non-zero minimum step size the robot can never reach the actual minimum position, instead the robot will cycle around the local minimum. This is illustrated in Figure 11.8, where the physical set-up is shown on top and the cycling resulting from the non-zero step moves is illustrated below. Robot dynamics, servo-controller, and sensor inaccuracies could have a similar effect. The time delays are typically a result of the computation times of the active component.

Consider, for example, the following case. The virtual sensor generates a virtual potential field to the active component. The active component continuously computes a position command, based on the current position of the

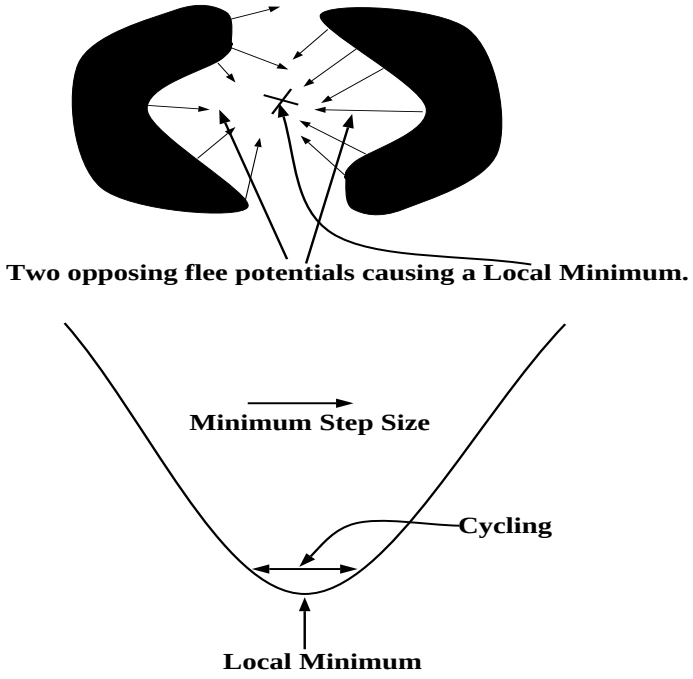


Figure 11.8: Robot between two obstacles with flee potentials. Non-zero minimum sized moves results in cycling around the resulting minimum.

robot. This position command could e.g. be generated as a position step added to the current position.

When the active component generates a command based on the current position, time delays and robot dynamics will cause the robot to overshoot or miss the generated position commands. If the new position command is generated by adding a small position or velocity vector to the current position which supposedly would take the robot closer to a desired minimum, the continuous misreadings of the current position (due to sampling delays) and other dynamic effects would most likely cause cycling around the minimum instead of convergence to the minimum.

By preplanning a path in the given potential field, this problem is eliminated, and we can still achieve the desired result. In this case the feedback loop to the active component is entirely eliminated. This is illustrated in Figure 11.9, which corresponds to the solution of the problem illustrated in Figure 11.8. The path to the local minimum is preplanned and generated to the underlying servo controller, instead of doing on-line "find the next point" calculations.

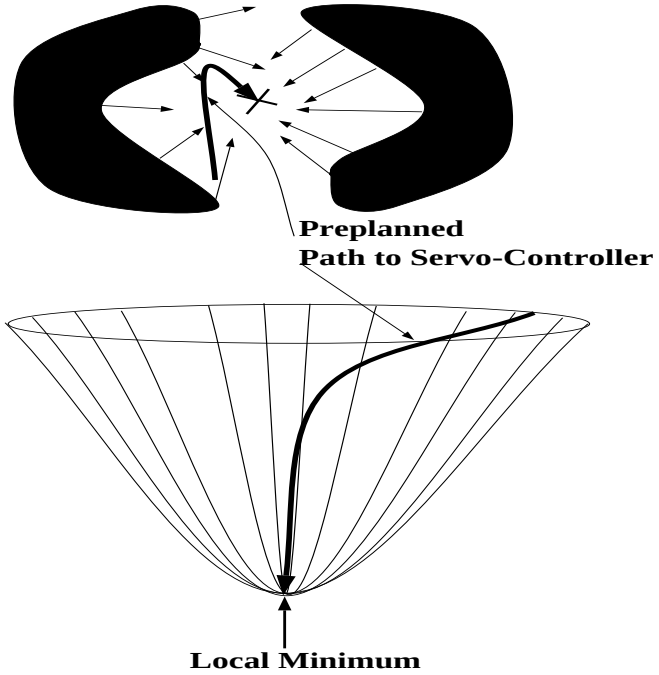


Figure 11.9: Robot between two obstacles with flee potentials. A preplanned path to the minimum, instead of on-line step-wise "next point" calculations.

An example of this situation occured during my experiments with a work-space-based potential function in the context of the sonar-based world mapping system. The commands to the servo-controller were generated on-line from a

potential function, resulting in cycling around local minima. Another type of problem occurred in the context of the flee reflexes. The velocity commands were originally derived on-line from the potential function generated around the moving obstacle. Due to servo-controller inaccuracies and time-delays the robot did not move strictly along the flee-direction, and deviated significantly from the straight y-axis path towards the end of the flee-response. When a local minimum appeared in the flee potential this also resulted in chattering.

To avoid these kinds of stability and cycling problems the following actions could be taken :

1. It might be possible to entirely eliminate this feedback loop. This was the action taken in the case of the flee reflexes. If the active component is relying on a pre-planned path there is no need to inform the active component about the current robot state. Doing this seems to imply that the active component would be operating open loop. This is usually not true when the virtual sensor, which is connected to the active component, still is a part of a feedback loop.

2. If it is impossible to eliminate the feedback loop containing the active component, make sure that the active component and the feedback loop is not unfavorably dependent on robot dynamics and time delays. This situation can be analyzed by applying Lyapunov functions, or the invariant-set theorem. If it is hard to apply control analysis, the effect of time-delays

and robot dynamics on the active component must be estimated in a conservative way, or dealt with in an empirical approach.

11.2.2 Instability and Cycling in the Virtual Sensor Feedback Loop

The cycling and instability problems which are generated by the virtual sensor are usually different in nature from the ones generated by the active component. The events in the virtual sensor feedback loop are usually of a smaller bandwidth, and often of a discrete nature. One example of a virtual sensor instability is that the reflexes are instantly turned back on again after they have been turned off, because the reflex off state causes the reflex to be turned back on. Another is that the virtual sensor is unable to decide whether the reflex should be on or off due to servo-controller or position sensor inaccuracies, or time-delays.

An example of this situation is the following: A flee reflex generates a flee response which brings the robot outside the flee potential field, where the reflex is turned off. When the robot exits the potential field, another module takes control of the robot. This module attempts to move the robot back into the flee potential field. Despite the fact that the flee reflex has precedence over this module this situation is likely to cause cycling. Time delays, robot dynamics, and servo-controller inaccuracies will cause the robot to reenter the

flee potential with some speed and distance before the flee potential is turned back on. This will generate a cycling pattern.

In this case we need to reconstruct the virtual sensor and the feedback loop so that they become robust with respect to robot dynamics and time delays. There are a few different ways of doing that, for example :

1. Make the turn-on conditions a strict subset of the turn-off conditions. In other words apply hysteresis to the reflex. Simple triggered reflexes are examples of reflexes with hysteresis. This is illustrated in Figure 10.10. The hysteresis region must be wider than the combined effect of the time-delays, servo controller inaccuracies, and the robot dynamics (brake-distance at expected speed). The block-emulating flee reflex is an example of a simple triggered repulsor reflex.

2. Make it impossible for the reflex to be reactivated under static conditions after it has been turned off. In other words, make sure that the goal of the reflex represents an invariant set. This can be done by replacing the turn off state with an "after the turn off" reflex. An example of this is the hold reflex for the block-emulating flee reflex. The hold-reflex, holds the robot in its position as long as commands from other modules could cause the robot to reenter the flee-potential field. In the case of the wall-emulating flee reflex, the reflexive command filter guarantees that the robot cannot reenter the flee-potential field. This way the reflexive

command filter provides the wall-emulating flee reflex with an invariant set, the free-space.

3. Do not allow the robot state to cause large, abrupt variations in virtual sensor output. An example of this is a reflex which causes a full-speed flee reaction if the robot is confined within a region in the robot state space, and is inactive everywhere else. The sharp boundary in the state space can be replaced by a region where the flee response gradually disappears.

11.3 Instability and Cycling Due To Unsuccessfully Integrated Modules in Real-Time

In this case bad dynamic interaction between two or more otherwise stable modules causes system instability. An example of this is if the interaction between the reflexive command filter, and the augmented task-space-based guiding potential functions did not comply with requirement 10.1.

This system has two control modes:

1. The reflexive command filter mode: Higher level commands in form of goals are given directly to the reflexive command filter.

2. The augmented task-space-based guiding potential function mode: The

augmented task-space-based guiding potential function[2] generates commands which bring the robot around local obstacles.

The system consisting of the reflexive command filter and an augmented task-space-based guiding potential function is shown in Figure 11.10.

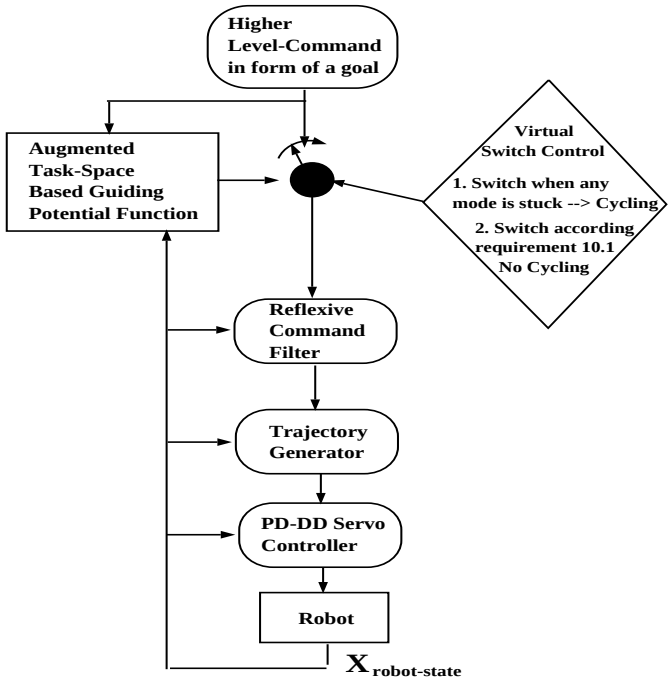


Figure 11.10: This system consists of an augmented task-space-based guiding potential function and a reflexive command filter. The switch control strategy must be chosen carefully.

The original control strategy for the interaction between the reflexive command filter and the augmented task-space-based guiding potential functions was :

1. When reflexive command filter fails to bring the robot to the goal, apply

[2]The augmented task-space coordinates used were: the three work-space coordinates for the wrist of the RRC robot, and the angle between the elbow of the RRC robot and the vertical plane through the shoulder and the wrist of the RRC robot

augmented task-space-based guiding potential function.

2. When the augmented task-space-based guiding potential function has brought the robot as far as it can, and the goal has still not been achieved, give the control back to the reflexive command filter.

In other words this control strategy can be summarized as, when one control mode fails, switch control mode. This control strategy resulted in cycling for the following reason. When the reflexive command filter failed to bring the robot to the goal due to an obstacle, the augmented task-space-based guiding potential function attempted to bring the robot closer to the goal measured in augmented task-space coordinates. However, in certain configurations such moves result in increasing the distance, as measured in joint-coordinates. Under the reflexive command filter control mode, the robot might be brought back to the configuration where the augmented task-space-based guiding potential function took over. This situation is illustrated in Figure 11.11.

The solution to this problem was to design a switch control method so that it complied with requirement 10.1. The new switch control strategy was: when one control mode fails, switch control mode if this is compatible with requirement 10.1. To deal with similar situations, requirement 10.1 should be applied. If that cannot be done easily, the use of multiple Lyapunov functions, or multiple Quasi Lagrange functions might be of help.

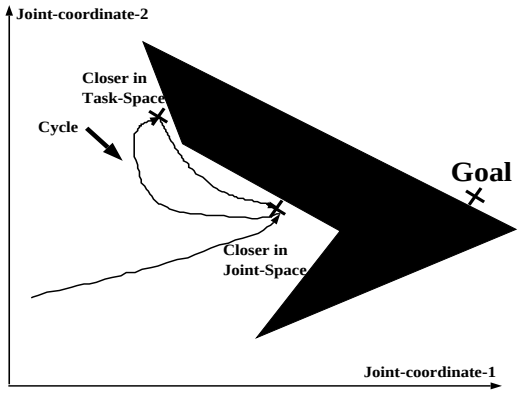


Figure 11.11: The control mode fails, switch control mode strategy might lead to cycling if the two control modes are described by different energy functions.

11.4 Instability and Cycling Due To Unsuccessfully Integrated Modules in Non-Real-Time

In this section, the problems related to unfavorable non-real-time interaction among different modules is discussed. The module interaction problems discussed here are similar to the ones discussed in Section 11.3 except that the interaction between the modules is not directly time related. Only the order in which events happens matters. In other words the modules in the system are incompatible with each other, regardless of dynamic interaction. For this reason the corresponding analysis involves logic, or discrete diagrams, rather than continuous energy measurements like Lyapunov functions.

Examples of this of type of problem are:

1. Suppose two reflexes $R_{attract}$ and $R_{repulse}$ have overlapping activation regions. Reflex $R_{attract}$ attempts to move the robot closer to an interesting obstacle $\mathcal{O}$, at the same time as reflex $R_{repulse}$ attempts to keep the robot away from this obstacle. Reflex $R_{repulse}$ is only triggered when the robot is close to the obstacle and generates thereafter an elongated "flee response". Further, reflex $R_{repulse}$ has precedence over reflex $R_{attract}$. When reflex $R_{attract}$ moves the robot towards obstacle $\mathcal{O}$, reflex $R_{repulse}$ is triggered, and the robot retracts away from the obstacle. When reflex $R_{repulse}$ has moved the robot to a safe distance from obstacle $\mathcal{O}$, reflex $R_{repulse}$ is turned off, and reflex $R_{attract}$ is turned on. Reflex $R_{attract}$ will move the robot closer to obstacle $\mathcal{O}$ only for reflex $R_{repulse}$ to be reinitiated. This interaction between reflex $R_{attract}$ and reflex $R_{repulse}$ results in cycling. In other words the way reflex $R_{attract}$ and reflex $R_{repulse}$ are constructed makes them incompatible with each other. The basic problem in this example is that the boundary of $R_{repulse}$ is not compatible with $R_{attract}$. How to resolve this is described in Section 10.5.

2. The Make-Map module in the sonar-based world mapping system module fails to generate temporary obstacle maps for recently investigated obstacles. When Investigate is done, the robot will remove itself from the obstacle in an attempt to continue investigating the rest of the workspace. However, when no map has been created for the obstacle, the

obstacle will be rediscovered by the sonar-based world mapping system as a new inconsistency and the investigation of the obstacle will start all over again. The robot will therefore engage in cycling. The cycling pattern is not a limit cycle; it is not even a quasi periodic cycle, nor a result of a strange attractor. In this case the cycling pattern is determined by constantly changing and unpredictable sensor data. How to analyze this situation is described in Section 10.6.

3. A world mapping system module A finds a new obstacle, while a second module B subsequently determines that the obstacle does not exist, and the obstacle is removed from the map. Module A finds the obstacle again while module B removes it from the map once again, in a repetitive cycle. In this case, two modules cannot agree on the existence of an obstacle, and the system is not structured to solve the conflict.

To ensure stability, the modules must be made compatible with each other. In sections 10.5 and 10.6 methods for determining inter-module stability are described. If the modules are simple or semi-simple reflexes in a non-hierarchical structure, the methods described in 10.5 can be utilized. In the more general case, it might be possible to define a progress measurement function, as in the case for the sonar-based world mapping system.

11.5 Design Advice Regarding Reflexive or Behavioral Systems.

In this section I will first give an overview of the types of modules that a behavioral robot system can consists of. When constructing a behavioral robot system it is first necessary to determine what kind of modules are needed in the system and what the exact function of these modules should be. When this is done the modules and the interaction between the modules can be analyzed using the techniques described in Chapter 10. The system can thereafter be built and tested in a step by step procedure. The next subsection gives an overview of the modules a behavioral robot system may consist of.

11.5.1 Possible Modules In A Behavioral Robot System.

The robot system architecture which will be discussed in this section is the structure shown in Figure 11.1. The types of modules this structure consist of are :

1. Reflexive actions which are by definition implemented in parallel with the control structure, and which compete for control of the robot. Like reflexes, they are simple and directly tied to sensory input. The following are examples of reflexive actions:

(a) Simple containment reflexes are active in a specific region of the state space, and all commands generated by the reflex are confined to this region. An example of this is the hold reflex.

(b) Simple repulsor reflexes are active in a specific region of the state space, but push the robot outside this region.

(c) Triggered reflexes are activated if the robot enters a certain region of the state space, called the trigger region, and remain active as long as the robot remains in a certain region of the state space called the activation region. Triggered containment reflexes generate commands which keep the robot inside the activation region. Triggered repulsor reflexes bring the robot outside the activation region. An example of a triggered repulsor reflex is the flee reflex.

(d) Internal situational reflexes are reflexes which detects certain conditions within the robot system, like which other modules are active, how fast the robot is moving, etc., and instantly generate the appropriate response. The halt-reflex is an internal situational reflex. Depending on which module is active, and the confidence threshold level, it generates a temporary halt.

(e) External situational reflexes are reflexes which detect certain external conditions, like an approaching obstacle, an approaching human, a loud noise, etc., and instantly generate the appropriate response.

(f) One-shot reflexes are reflexes which detect a certain internal or external condition and generate a temporary high-priority response which disappears even if the stimulus remains. An example of this type of reflex is the startle reflex.

2. Reflexive command filters which are by definition implemented in series with the control structure. All higher-level commands must pass through a reflexive command filter. An example of a reflexive command filter is the reflexive command filter for obstacle avoidance described in this thesis. The reflexive command filter for obstacle avoidance protected the robot from collision that would result if erroneous higher-level commands were carried out. In other words, the reflexive command filter for obstacle avoidance prevents collisions resulting from software errors. Other types of reflexive command filters could prevent obviously incorrect torque commands, illogical or forbidden actions, or dangerous forces from being exerted.

3. Fixed action patterns which are extended, largely stereotyped responses to sensory stimuli. Fixed action patterns are similar to reflexes in the sense that they are simple responses. However, they are extended responses which generate an action pattern which continues to exist long after the stimulus has dissapeared. An example is Investigate in the sonar-based world mapping system. Investigate moves the robot tool flange up

and down and around a found obstacle in a largely stereotyped manner. It should be noted that fixed action patterns do not contain any complex algorithms, planning, or advanced perception.

4. Higher level modules containing complex algorithms, planning, or advanced perception. A higher-level module can exists in the form of command filters. An example of this is a path-planner which generates paths to underlying levels from all higher-level commands. A path-planner can also be a module which is activated only in certain situations.

11.5.2 A Step By Step Procedure For Constructing A Robotic Behavioral System

The following is a suggested procedure for constructing a behavioral system, assuming that the servo-controller already exists:

1. State the purpose of your system, and all functions you would like your system to have. Generate a modular system similar to the one shown in Figure 11.1, which would theoretically achieve the purpose of the system.

2. Identify the detailed interaction among the modules, and define the function of each module's virtual sensor and active component.

3. Identify the detailed function of each module, and its corresponding virtual sensor and active component. Analyze the stability of each module using Lyapunov functions, the invariant set theorem, Quasi Lagrange functions, or an other method. If a module cannot be proven to be stable, redesign it.

4. Analyze the stability and performance of the feedback loop containing the active component of each module, the servo-controller, trajectory generator and the robot. This can be done by utilizing Lyapunov functions, the invariant set theorem, Quasi Lagrange functions, or any other method. If this feedback loop cannot be proven to be stable, redesign it.

5. Analyze the stability and performance of the feedback loop containing the virtual sensor of each module, the servo-controller, trajectory generator and the robot. This can be done by utilizing Lyapunov functions, the invariant set theorem, Quasi Lagrange functions, or any other method. If this feedback loop cannot be proven to be stable, redesign it.

6. Using the argument concerning inner and outer loops in Section 11.2, determine wheather it is plausible that the combined feedback loops are stable. If not analyze the combined system consisting of both feedback loops.

7. Analyze the interaction among all modules at the same level using multiple Lyapunov functions, multiple Quasi Lagrange functions, or progress measurement functions. If the modules are simple-containment reflexes, triggered-containment reflexes, or simple-repulsor reflexes, use the methodology developed in Section 10.5 to analyze the module interaction. Redesign the module interaction if the system cannot be shown to be stable or to perform well.

8. Build each individual module, test it by itself, and in a feedback loop containing the servo-controller and the robot. Determine that the module is stable and performs well both in feedback and by itself. If not, debug the module and if necessary redesign it.

9. Integrate the system, test it, and determine that the system is stable and performs well. If the system does not perform well, debug or redesign the interaction between the modules, and if necessary, redesign the entire system. If the latter has to be done you might have to restart the design procedure at step 1.

If each step is done very carefully and the stability analysis in step 3—7 is properly done, it should not be necessary to restart the design procedure when step 9 has been reached.

11.6 Final Conclusions and Overview for Chapter 11

This chapter gave an overview of the stability and cycling problems I encountered during my experiments. This chapter provided qualitative advice on how to construct reflex modules, and how to design systems containing interacting behavioral modules.

The stability and performance problems I encountered during my experiments were of the following types:

1. One module, or a set of interacting modules, is badly designed and therefore behaves unstable in open loop. An illustratation of the locality of these types of problems in the pine-tree structure is made in Figure 11.2. The reasons why individual behavioral modules are unstable are, for example, that the basic module design is flawed, or that the computer program implementing the module contains bugs. The stability of a single module can be analyzed using Lyapunov functions, the local set invariant theorem, or Quasi Lagrange functions.

2. One or more modules are unstable in their respective feedback loop. This is illustrated in Figure 11.3. Instability can appear due to the active components dependency on the robot state, as well as the virtual sensors dependency on the robot state. This situation can be analyzed by

analyzing the individual feedback loops, or if possible, the combined system. If the difference in bandwidth between successive loop closures is an order of a magnitude, it is likely that the two feedback loop systems can be analyzed separately. If there is an instability problem generated in the feedback loop from the robot output to the active component of the reflex module, this can usually be solved by entirely eliminating the feedback loop. If there is an instability problem generated in the feedback loop from the robot output to the virtual sensor this can be solved by regulating how the virtual sensor is activated.

3. Real-time interaction problems among the modules. This is illustrated in Figure 11.4. A solution to these problems is to design a switch control method which complies with requirement 10.1.

4. Non-real-time interaction problems among the modules. This could also be illustrated by Figure 11.4. If the modules are simple or semi-simple reflexes in a non-hierarchical structure the methods described in 10.5 can be utilized to analyze this situation. It might also be possible to define a progress measurement function which is useful for analysis.

Section 11.5 gave an overview of a formal procedure for constructing behavioral systems. This procedure is very general and does not provide any extensive revealing information by itself. However, in conjunction with the

methods discussed and given in Chapter 10 and the experience expressed in this chapter this procedure can serve as a guide-line for future system design.

Chapter 12

Conclusions and Suggestions for Further Work

This chapter will give a very brief overview of the content of this thesis, discuss the results with respect to the thesis objective, and present a few conclsuions. This chapter will also present a few suggestions for further work in reflex-like control systems. It is my hope that others will find find this area of research interesting and valuable and pursue similar research.

12.1 Summary and conclusions

This thesis has presented a survey of my work in reflex-based robot control. The appeal to reflex-like, stimulus-response behaviors is that a machine thus controlled can be more robust with respect to unanticipated events. Most commonly, unanticipated situations are considered from the context of unpredictable environment stimuli. Another likely source of unanticipated events is internal—due to embedded software errors in the controller itself. Using reflexive behaviors, we can "trap" potentially destructive errors, e.g. with

reflexive collision avoidance.

The premise of reflex control is that one should create low-level behaviors that are ordinarily inactive or virtually "transparent" with respect to higher-level commands. However, these behaviors should trigger rapidly to perform protective responses when there is indication that the host machine is in danger. By design, the reflex layer is not "intelligent" by any reasonable measure. Rather, reflex actions consist of fixed relationships between stimulus and response; no learning or deductive analysis takes place. Nonetheless, the response executed by a reflex is typically appropriate to an emergency condition. Thus, reflexes trade off sophistication of analysis for speed of response.

Using a "reflexive command filter" the robot is prevented from colliding with itself (including joint limits) or with known objects in its environment. This behavior is like a filter, in that its protective action occurs by refusing to approve commands which advance the robot into danger. The addition of the reflexive command filter permits appropriate and timely protective responses, and this additional robustness is obtained without sacrificing the performance of the rest of the system.

It was found that the path-planning task was simplified by the assumption that the lower-level reflexive command filter would guarantee collision avoidance. With this assumption, the path planner could consider lower-resolution

maps and coarser approximations of robot dynamics, leaving execution refinements to the reflexes.

In addition, I presented more proactive low-level behaviors, including the "startle" and "retraction" and "hold" reflexes, in the case of multi-arm collision avoidance. The existence of the flee reflex permitted a dramatically simpler implementation of higher-level cooperative controls. Both the wall-emulating and block emulating flee reflexes executes rapidly and are functionally simple, and are therefore practical implementations of multi-arm collision avoidance.

This thesis also discussed fixed action patterns, which are behaviors similar to reflexes in the sense that they were low-level, but different in the sense that the corresponding output was not directly tied to the input. Reflex modules, and fixed action patterns were used to build a system which performed autonomous map building using sonar sensors. This demonstrates that reflex-like control and fixed action patterns can be used to build complex autonomous systems which performs complex tasks. The resulting system was modular, flexible, and could be incrementally expanded which is another characteristica of reflex-like or behavioral systems.

All systems and modules described in this thesis were implemented and tested on an industrial manipulator. Each module represented an individual behavior, and was shown to be compatible with command generation from higher-levels. Even though it is possible to build complex systems using reflex

modules and other low-level behavioral modules, the main purpose of the reflex approach is not to replace sophisticated planners or other higher-levels of cognition. Rather, the reflexes augment a complex system by enhancing its robustness with respect to unanticipated events; further, low-level reflexes simplifies the task of designing higher levels of cognition. The existence of additional reflex behaviors could have similar simplifying effects on the design of higher-level controls.

In this thesis the usefulness of several standard methods for stability and performance analysis was discussed. Further, this thesis introduced a few new methods specifically designed for analysis of behavioral systems. The concept of stability in the sense of Lyapunov was discussed and implemented on the static command filter and the guiding potential functions. The interaction between the static command filter and the guiding potential functions was analyzed using global Lyapunov functions, and multiple Lyapunov functions. The invariant set theorem was succesfully used to show that the wall-emulating flee reflexes, and the block-emulating flee reflexes resulted in the robot converging to free-space.

The concept of Quasi Lagrange stability was introduced so that the flee reflexes could be analyzed in terms of stability. Stability and convergence analysis applied to the output vectors of higher-level modules was validated using the concept of command-tolerant stability. Methods for analyzing the

the interactions between simple-containment, and triggered-containment and repulsor reflexes were introduced and discussed. In section 10.6, progress measurement functions were used to analyze the performance of the sonar-based world mapping system.

The concepts discussed and introduced in this thesis were applied to the reflexes and behavioral modules I constructed. It was demonstrated that Lyapunov functions, multiple Lyapunov functions, the set-invariant theorem, and (Quasi) Lagrange stability concepts, are useful concepts for real time analysis of behavioral systems. The new concepts introduced were also shown to be useful. The stability and performance problems I encountered during my experiments were discussed, and advice regarding how to avoid and deal with such problems were given. A general design method for behavioral systems was also presented.

This thesis presented different implementations of reflex-like and behavioral control methods. These implementations demonstrated that reflex control is a useful concept for achieving robot system safety, reliability and robustness. They also demonstrate that reflex control can be used as an important component of complex autonomous systems. This thesis also discussed how reflexes should be constructed to be useful, and how stability or cycling problems could be avoided when adding reflex modules to the system.

12.2 Further Work

One obvious extension to the work presented here is the improvement, or the extension of the modules presented here. Examples of such improvements are:

1. In the reflexive command filter for obstacle avoidance, add a submodule to the C-space inspector which generates local C-space, using the template method and a list of nearby obstacles and their obstacle features. This can be used for higher resolution C-space of in higher dimensions.

2. In the reflexive command filter for obstacle avoidance, add a submodule to the C-space inspector which generates local C-space, using a Jacobian-based method.

3. Apply the reflexive command filter to work-space using distance measurements instead of using C-space. This would especially be useful for highly redundant robots.

4. Use the reflexive command filter in conjunction with better path planners which are truly subgoal based, irrespective of the dimensionality of the C-space.

5. Improve the flee reflexes, or invent new flee reflex concepts.

6. Extend, redesign, or improve the sonar-based world mapping system to become even more general, efficient, and perhaps more modular.

7. Improve the sonar-data-based map-generation process to conform to the most efficient and complete methods in existence.

8. Make a template-based C-space generator which is more efficient, or includes more obstacle features, or operates in higher dimensionality.

9. Make a general template-based C-space generator for arbitrary robots. The C-space generator first generates the necessary feature databases after it has been given information about the robot. The necessary robot information the C-space generator needs is a CAD/CAM description of the robot links and a description of the robot's configuration (DH parameters).

Another obvious extension to the work presented here is the development of additional reflexive and behavioral modules for other purposes than obstacle avoidance and sonar-based world mapping. Examples of such potential modules are:

1. Reflexive command filters for compliant control. These reflexive command filters would remove all commands that would result in excessive forces, unfeasible stiffness and damping, or forces and velocities which are undesirable in the context of the current system and environment.

2. Reflexive command filters for velocity, torque, voltage or current commands that could result in system damage. Such a filter would check

that system software (a servo-controller) never generates, for example, PWM-signals that could destroy a servo-amplifier.

3. Reflexive command filters which would protect the system from commands that obviously are incorrect in a certain context. For example, at start up a huge step command is probably a result of an incorrect position command, or a position command which incorrectly remains from an earlier batch. Another example is a reflexive command filter which prevents the robot from picking up a piece in a bin while moving at a high speed. If a higher module generates this type of command it must result from a software error.

4. Reflexive actions, or fixed action patterns that are submodules in certain action schemes for robot walking, robot gripping, robot climbing, robot jumping, etc.

5. Reflexive actions which act as transition modules when switching control mode in the case of, for example, a walking robot which slips, steps in a hole, etc..

6. Reflexive actions for numerous special tasks like:

   (a) Autonomous car, or mobile robot path following.

   (b) Autonomous car, mobile robot, or industrial robot collision avoidance.

(c) Unexpected events like a walking robot slipping, or stepping in a hole, or a mobile robot reaching a steep edge or an overly narrow corridor.

Improvements and generalizations can also be made to the analysis presented in Chapter 10 and 11. These improvements include:

1. The application of the methods described in Chapter 10 and 11 to a larger set of practical examples.

2. A more rigorous or general description of the methods applied in Chapter 10 and 11.

3. The implementation of standard methods not mentioned in Chapter 10 and 11 for analysis of reflexive and behavioral systems.

4. The development of new methods useful for analyzing reflexive and behavioral systems.

5. The development of a more detailed and practical design scheme than the one described in section 11.5.

Most research efforts within robotics have been concentrated on:

- Servo-controllers, trajectory generators, kinematics, dynamics, calibration, position-control, velocity-control, force-control, compliant-control, and other low-level applications. This research has been beneficial to

the industry which has been able to improve the speed and accuracy of industrial robots.

- Path-planners, perception modules, map-generators, C-space generation methods, task-planning, artificial intelligence, and other high-level applications. This research has been beneficial to certain applications within the research community and special applications like, waste cleanup using robots, space exploration, etc.. However, this type of research has to large extent failed to increase the autonomy and employability of robots.

In this context, I believe that reflexes and other low-level behaviors represent the forgotten middle layer. This thesis has demonstrated that this middle layer could serve as a means to connect high-level controls with low-level controls. I believe that in the long run, this will result in the construction of advanced, flexible, robust, and autonomous robot system that, to a much larger extent, could replace human laborers, do things which currently cannot be done, and assist in solving problems which currently do not have a solution.

Appendix A

Kinematic model for the RRC

This appendix describes the kinematics of the RRC robot. The joint coordinate systems I chose are given in Figure A.1[1], and the corresponding DH-parameters are shown Table A.1. The values of θ_i in the configuration shown are given in Table A.2.

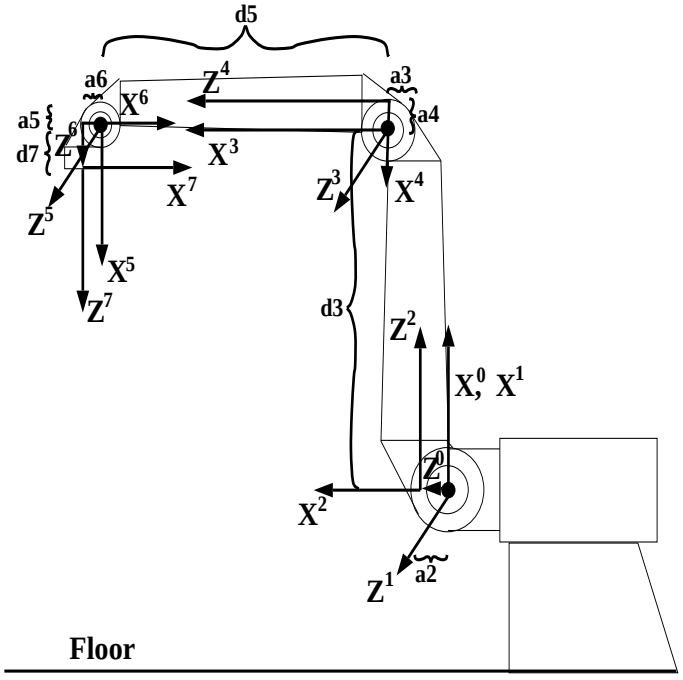


Figure A.1: The joint coordinate systems of the RRC robot and the corresponding parameters.

[1]Also given in section 2.2

Joint	θ_i	α_i	d_i	a_i
1	θ_1	$90°$	0	0
2	θ_2	$90°$	0	$a2$
3	θ_3	$-90°$	$d3$	$a3$
4	θ_4	$90°$	0	$a4$

Table A.1: DH-parameters for the RRC robot.

$\theta_1 = 0°$
$\theta_2 = 90°$
$\theta_3 = 0°$
$\theta_4 = 90°$

Table A.2: The home angles of the RRC robot

Equation A.1 below give the transformation matrix from the base frame to the world frame, and equations A.2, A.3, A.4, and A.5 gives the transformation matrices for the first four joint frames. The resulting joint-4 frame to base frame transformation matrix is given in equation A.6, and the resulting joint-4 frame to world frame transformation matrix is given in equation A.7. Equation A.8 shows the transformation matrix from joint-6 to joint-4. The transformation matrices shown are used to compute configuration space, robot tool and elbow locations, potential functions, and the configurations corresponding to desired sonar beam directions.

$$ {}^{W}A_0 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.1} $$

$$ {}^{0}A_1 = \begin{pmatrix} cos(\theta_1) & 0 & sin(\theta_1) & 0 \\ sin(\theta_1) & 0 & -cos(\theta_1) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.2} $$

$$ {}^{1}A_2 = \begin{pmatrix} cos(\theta_2) & 0 & sin(\theta_2) & a2cos(\theta_2) \\ sin(\theta_2) & 0 & -cos(\theta_2) & a2sin(\theta_2) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.3} $$

$$ {}^{2}A_3 = \begin{pmatrix} cos(\theta_3) & 0 & -sin(\theta_3) & a3cos(\theta_3) \\ sin(\theta_3) & 0 & cos(\theta_3) & a3sin(\theta_3) \\ 0 & -1 & 0 & d3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.4} $$

$$ {}^{3}A_4 = \begin{pmatrix} cos(\theta_4) & 0 & sin(\theta_4) & a4cos(\theta_4) \\ sin(\theta_4) & 0 & -cos(\theta_4) & a4sin(\theta_4) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.5} $$

$$
{}^{0}A_4 =
$$

$$
\begin{pmatrix}
\begin{array}{l} cos(\theta_1)cos(\theta_2)cos(\theta_3)cos(\theta_4) \\ +sin(\theta_1)sin(\theta_3)cos(\theta_4) \\ -cos(\theta_1)sin(\theta_2)sin(\theta_4) \end{array} &
\begin{array}{l} sin(\theta_1)cos(\theta_3)- \\ cos(\theta_1)cos(\theta_2)sin(\theta_3) \end{array} &
\begin{array}{l} cos(\theta_1)cos(\theta_2)cos(\theta_3)sin(\theta_4) \\ +sin(\theta_1)sin(\theta_3)sin(\theta_4)+ \\ cos(\theta_1)sin(\theta_2)cos(\theta_4) \end{array} &
\begin{array}{l} a4(sin(\theta_1)sin(\theta_3)cos(\theta_4)+ \\ cos(\theta_1)cos(\theta_2)cos(\theta_3)cos(\theta_4) \\ -cos(\theta_1)sin(\theta_2)sin(\theta_4))+ \\ a3(sin(\theta_1)sin(\theta_3)+ \\ cos(\theta_1)cos(\theta_2)cos(\theta_3))+ \\ d3cos(\theta_1)sin(\theta_2)+ \\ a2cos(\theta_1)cos(\theta_2) \end{array} \\
\begin{array}{l} sin(\theta_1)cos(\theta_2)cos(\theta_3)cos(\theta_4) \\ -cos(\theta_1)sin(\theta_3)cos(\theta_4)- \\ sin(\theta_1)sin(\theta_2)sin(\theta_4) \end{array} &
\begin{array}{l} -sin(\theta_1)cos(\theta_2)sin(\theta_3) \\ -cos(\theta_1)cos(\theta_3) \end{array} &
\begin{array}{l} sin(\theta_1)cos(\theta_2)cos(\theta_3)sin(\theta_4) \\ -cos(\theta_1)sin(\theta_3)sin(\theta_4) \\ +sin(\theta_1)sin(\theta_2)cos(\theta_4) \end{array} &
\begin{array}{l} a4(-cos(\theta_1)sin(\theta_3)cos(\theta_4)+ \\ sin(\theta_1)cos(\theta_2)cos(\theta_3)cos(\theta_4) \\ -sin(\theta_1)sin(\theta_2)sin(\theta_4))- \\ a3(cos(\theta_1)sin(\theta_3)- \\ sin(\theta_1)cos(\theta_2)cos(\theta_3))+ \\ d3sin(\theta_1)sin(\theta_2)+ \\ a2sin(\theta_1)cos(\theta_2) \end{array} \\
\begin{array}{l} sin(\theta_2)cos(\theta_3)cos(\theta_4) \\ +cos(\theta_2)sin(\theta_4) \end{array} &
-sin(\theta_2)sin(\theta_3) &
\begin{array}{l} sin(\theta_2)cos(\theta_3)sin(\theta_4) \\ -cos(\theta_2)cos(\theta_4) \end{array} &
\begin{array}{l} a4(sin(\theta_2)cos(\theta_3)cos(\theta_4) \\ +cos(\theta_2)sin(\theta_4))+ \\ a3sin(\theta_2)cos(\theta_3)- \\ d3cos(\theta_2) + a2sin(\theta_2) \end{array} \\
0 & 0 & 0 & 1
\end{pmatrix}
\tag{A.6}
$$

$$
{}^{W}A_4 =
$$

$$
\begin{pmatrix}
sin(\theta_2)cos(\theta_3)cos(\theta_4) & -sin(\theta_2)sin(\theta_3) & sin(\theta_2)cos(\theta_3)sin(\theta_4) & a4(sin(\theta_2)cos(\theta_3)cos(\theta_4) \\
+cos(\theta_2)sin(\theta_4) & & -cos(\theta_2)cos(\theta_4) & +cos(\theta_2)sin(\theta_4))+ \\
 & & & a3sin(\theta_2)cos(\theta_3)- \\
 & & & d3cos(\theta_2) + a2sin(\theta_2) \\
 & & & \\
cos(\theta_1)sin(\theta_3)cos(\theta_4)- & sin(\theta_1)cos(\theta_2)sin(\theta_3) & cos(\theta_1)sin(\theta_3)sin(\theta_4)- & a4(cos(\theta_1)sin(\theta_3)cos(\theta_4)- \\
sin(\theta_1)cos(\theta_2)cos(\theta_3)cos(\theta_4) & +cos(\theta_1)cos(\theta_3) & sin(\theta_1)cos(\theta_2)cos(\theta_3)sin(\theta_4) & sin(\theta_1)cos(\theta_2)cos(\theta_3)cos(\theta_4) \\
+sin(\theta_1)sin(\theta_2)sin(\theta_4) & & -sin(\theta_1)sin(\theta_2)cos(\theta_4) & +sin(\theta_1)sin(\theta_2)sin(\theta_4))+ \\
 & & & a3(cos(\theta_1)sin(\theta_3)- \\
 & & & sin(\theta_1)cos(\theta_2)cos(\theta_3))- \\
 & & & d3sin(\theta_1)sin(\theta_2)- \\
 & & & a2sin(\theta_1)cos(\theta_2) \\
 & & & \\
cos(\theta_1)cos(\theta_2)cos(\theta_3)cos(\theta_4) & sin(\theta_1)cos(\theta_3)- & cos(\theta_1)cos(\theta_2)cos(\theta_3)sin(\theta_4) & a4(cos(\theta_1)cos(\theta_2)cos(\theta_3)cos(\theta_4) \\
+sin(\theta_1)sin(\theta_3)cos(\theta_4) & cos(\theta_1)cos(\theta_2)sin(\theta_3) & +sin(\theta_1)sin(\theta_3)sin(\theta_4)+ & +sin(\theta_1)sin(\theta_3)cos(\theta_4)- \\
-cos(\theta_1)sin(\theta_2)sin(\theta_4) & & cos(\theta_1)sin(\theta_2)cos(\theta_4) & -cos(\theta_1)sin(\theta_2)sin(\theta_4))+ \\
 & & & a3(sin(\theta_1)sin(\theta_3)+ \\
 & & & cos(\theta_1)cos(\theta_2)cos(\theta_3))+ \\
 & & & d3cos(\theta_1)sin(\theta_2)+ \\
 & & & a2cos(\theta_1)cos(\theta_2) \\
 & & & \\
0 & 0 & 0 & 1
\end{pmatrix}
$$

(A.7)

$$
{}^{4}A_6 =
\begin{pmatrix}
cos(\theta_5)cos(\theta_6) & -sin(\theta_5) & cos(\theta_5)sin(\theta_6) & a6cos(\theta_5)cos(\theta_6) + a5sin(\theta_5) \\
sin(\theta_5)cos(\theta_6) & cos(\theta_5) & cos(\theta_5)sin(\theta_6) & a6sin(\theta_5)cos(\theta_6) + a5cos(\theta_5) \\
-sin(\theta_6) & 0 & cos(\theta_6) & d5 - a6sin(\theta_6) \\
0 & 0 & 0 & 1
\end{pmatrix}
$$

(A.8)

The Manipulator Jacobian $\delta\vec{x} = J\delta\vec{\theta}$, or $J = \frac{\delta x_i}{\delta \theta_j}$, is used for the flee reflexes described in Chapter 8 and the ultrasound-based world mapping system described in Chapter 9. The Jacobian is calculated with respect to points located in link-2 and link-4, in other words located in the upper and lower manipulator arm. The Jacobian matrices for link-2 and link-4 are given in Equations A.9 and A.10 respectively. $P2$ and $P4$ denotes points located on the Z_2 and the Z_4 axis respectively.

$$J^2 = \begin{pmatrix} 0 & P2sin(\theta_2) + a2cos(\theta_2) - P2cos(\theta_1)sin(\theta_2) - \\ & a2cos(\theta_1)cos(\theta_2) \\ \\ a2sin(\theta_1)sin(\theta_2) - P2sin(\theta_1)cos(\theta_2) & \\ \\ -P2sin(\theta_1)sin(\theta_2) - a2sin(\theta_1)cos(\theta_2) & P2cos(\theta_1)cos(\theta_2) - a2cos(\theta_1)sin(\theta_2) \end{pmatrix} \quad \text{(A.9)}$$

$$J^4 =$$

$$\begin{pmatrix}
0 & \begin{array}{c} P4(sin(\theta_2)cos(\theta_4)+ \\ cos(\theta_2)cos(\theta_3)sin(\theta_4)) \\ +a4(-sin(\theta_2 sin(\theta_4)+ \\ cos(\theta_2)cos(\theta_3)cos(\theta_4)) \\ +a3cos(\theta_2)cos(\theta_3)+ \\ a2cos(\theta_2)+d3sin(\theta_2) \end{array} & \begin{array}{c} -P4sin(\theta_2)sin(\theta_3)sin(\theta_4) \\ -a4sin(\theta_2)cos(\theta_3)cos(\theta_4) \\ -a3sin(\theta_2)sin(\theta_3) \end{array} & \begin{array}{c} P4(cos(\theta_2)sin(\theta_4)+ \\ sin(\theta_2)cos(\theta_3)cos(\theta_4)) \\ +a4(cos(\theta_2)cos(\theta_4)- \\ sin(\theta_2)cos(\theta_3)sin(\theta_4)) \end{array} \\
J^4_{2,1} & \begin{array}{c} -P4(sin(\theta_1)cos(\theta_2)cos(\theta_4)- \\ sin(\theta_1)sin(\theta_2)cos(\theta_3)sin(\theta_4)) \\ +a4(sin(\theta_1)cos(\theta_2)sin(\theta_4)+ \\ sin(\theta_1)sin(\theta_2)cos(\theta_3)cos(\theta_4)) \\ +a3sin(\theta_1)sin(\theta_2)cos(\theta_3)+ \\ a2sin(\theta_1)sin(\theta_2)- \\ d3cos(\theta_1)sin(\theta_2) \end{array} & \begin{array}{c} P4(sin(\theta_1)cos(\theta_2) \\ sin(\theta_3)sin(\theta_4)+ \\ cos(\theta_1)cos(\theta_3)sin(\theta_4)) \\ +a4(sin(\theta_1)cos(\theta_2) \\ sin(\theta_3)cos(\theta_4)+ \\ sin(\theta_1)cos(\theta_2)sin(\theta_4)) \\ +a3(cos(\theta_1)cos(\theta_3)+ \\ sin(\theta_1)cos(\theta_2)sin(\theta_3)) \end{array} & \begin{array}{c} P4(cos(\theta_1)sin(\theta_3)cos(\theta_4)+ \\ sin(\theta_1)sin(\theta_2)sin(\theta_4)- \\ sin(\theta_1)cos(\theta_2)cos(\theta_3)cos(\theta_4)) \\ +a4(sin(\theta_1)sin(\theta_2)cos(\theta_4)- \\ cos(\theta_1)sin(\theta_3)sin(\theta_4)+ \\ sin(\theta_1)cos(\theta_2)cos(\theta_3)sin(\theta_4)) \end{array} \\
J^4_{3,1} & \begin{array}{c} P4(cos(\theta_1)cos(\theta_2)cos(\theta_4)- \\ cos(\theta_1)sin(\theta_2)cos(\theta_3)sin(\theta_4)) \\ -a4(cos(\theta_1)cos(\theta_2)sin(\theta_4)+ \\ cos(\theta_1)sin(\theta_2)sin(\theta_3)cos(\theta_4)) \\ -a3cos(\theta_1)sin(\theta_2)cos(\theta_3)- \\ a2cos(\theta_1)sin(\theta_2)+ \\ d3cos(\theta_1)cos(\theta_2) \end{array} & \begin{array}{c} -P4(cos(\theta_1)cos(\theta_2) \\ sin(\theta_3)sin(\theta_4) \\ +sin(\theta_1)cos(\theta_3)sin(\theta_4)) \\ -a4((cos(\theta_1)cos(\theta_2) \\ sin(\theta_3)cos(\theta_4) \\ sin(\theta_1)cos(\theta_3)sin(\theta_4)) \\ +a3(sin(\theta_1)cos(\theta_3)- \\ cos(\theta_1)cos(\theta_2)sin(\theta_3)) \end{array} & \begin{array}{c} P4(sin(\theta_1)sin(\theta_3)cos(\theta_4)+ \\ cos(\theta_1)cos(\theta_2)cos(\theta_3)cos(\theta_4) \\ -cos(\theta_1)sin(\theta_2)sin(\theta_4))- \\ a4(sin(\theta_1)sin(\theta_3)sin(\theta_4)+ \\ cos(\theta_1)cos(\theta_2)cos(\theta_3)sin(\theta_4) \\ -cos(\theta_1)sin(\theta_2)cos(\theta_4)) \end{array} \\
T_1 & T_2 & T_3 & T_4
\end{pmatrix} \tag{A.10}$$

Where $J^4_{2,1}, J^4_{3,1}, T_1, T_2, T_3, T_4$ are given below.

$$
\begin{aligned}
J^4_{2,1} &= -P4(sin(\theta_1)sin(\theta_3)sin(\theta_4) + cos(\theta_1)sin(\theta_2)cos(\theta_4) + \\
&\quad cos(\theta_1)cos(\theta_2)cos(\theta_3)sin(\theta_4)) + a4(cos(\theta_1)sin(\theta_2)sin(\theta_4) - \\
&\quad cos(\theta_1)cos(\theta_2)cos(\theta_3)cos(\theta_4) - sin(\theta_1)sin(\theta_3)cos(\theta_4)) \\
&\quad -a3(sin(\theta_1)sin(\theta_3) + cos(\theta_1)cos(\theta_2)cos(\theta_3)) + \\
&\quad a2cos(\theta_1)cos(\theta_2)d3cos(\theta_1)sin(\theta_2)
\end{aligned}
$$

$$
\begin{aligned}
J^4_{3,1} &= P4(cos(\theta_1)sin(\theta_3)sin(\theta_4) - sin(\theta_1)sin(\theta_2)cos(\theta_4) - \\
&\quad sin(\theta_1)cos(\theta_2)cos(\theta_3)sin(\theta_4)) - a4(cos(\theta_1)sin(\theta_3)cos(\theta_4) + \\
&\quad sin(\theta_1)sin(\theta_2)sin(\theta_4) - sin(\theta_1)cos(\theta_2)cos(\theta_3)cos(\theta_4)) \\
&\quad +a3(sin(\theta_1)cos(\theta_2)cos(\theta_3) - cos(\theta_1)sin(\theta_3)) - \\
&\quad a2sin(\theta_1)cos(\theta_2) - d3sin(\theta_1)sin(\theta_2)
\end{aligned}
$$

$$
\begin{aligned}
T_1 &= \frac{(W_yE^1_x + W^1_yE_x - W_xE^1_y - W^1_xE_y)E_z + (W_yE_x - W_xE_y)E^1_z}{E_zE_z + (W_yE_x - W_xE_y)^2} \\
T_2 &= \frac{(W_yE^2_x + W^2_yE_x - W_xE^2_y - W^2_xE_y)E_z + (W_yE_x - W_xE_y)E^2_z}{E_zE_z + (W_yE_x - W_xE_y)^2} \\
T_3 &= \frac{(W_yE^3_x + W^3_yE_x - W_xE^3_y - W^3_xE_y)E_z + (W_yE_x - W_xE_y)E^3_z}{E_zE_z + (W_yE_x - W_xE_y)^2} \\
T_4 &= \frac{(W_yE^4_x + W^4_yE_x - W_xE^4_y - W^4_xE_y)E_z + (W_yE_x - W_xE_y)E^4_z}{E_zE_z + (W_yE_x - W_xE_y)^2}
\end{aligned}
$$

Where $\vec{W} = (W_x, W_y, W_z)$ is the location of the wrist (frame-5) with respect to the world coordinate system, and $\vec{E} = (E_x, E_y, E_z)$is the location of the

elbow (frame-3) with respect to the world coordinate system. W_x^i, W_y^i, W_z^i and E_x^i, E_y^i, E_z^i are the differentials of the wrist and elbow coordinates with respect to the joint angle θ_i. The world coordinates for the elbow and the wrist are listed below.

$$
\begin{aligned}
E_x &= a2cos(\theta_2) - d3cos(\theta_2) \qquad \text{(A.11)} \\
E_y &= -d3sin(\theta_1)sin(\theta_2) - a2sin(\theta_1)cos(\theta_2) \\
E_z &= d3cos(\theta_1)sin(\theta_2) + a2cos(\theta_1)cos(\theta_2) \\
W_x &= {}^{W}A_4(1,2)W_{z_4} + {}^{W}A_4(1,3) \\
W_y &= {}^{W}A_4(2,2)W_{z_4} + {}^{W}A_4(2,3) \\
W_z &= {}^{W}A_4(3,2)W_{z_4} + {}^{W}A_4(3,3) \\
E_x^1 &= 0 \\
E_x^2 &= -a2sin(\theta_2) - d3cos(\theta_2) \\
E_x^3 &= 0 \\
E_x^4 &= 0 \\
E_y^1 &= -d3cos(\theta_1)sin(\theta_2) - a2cos(\theta_1)cos(\theta_2) \\
E_y^2 &= -d3sin(\theta_1)cos(\theta_2) + a2sin(\theta_1)sin(\theta_2) \\
E_y^3 &= 0 \\
E_y^4 &= 0
\end{aligned}
$$

$$
\begin{aligned}
E_z^1 &= -d3sin(\theta_1)sin(\theta_2) - a2sin(\theta_1)cos(\theta_2) \\
E_z^2 &= d3cos(\theta_1)cos(\theta_2) + a2cos(\theta_1)sin(\theta_2) \\
E_z^3 &= 0 \\
E_z^4 &= 0 \\
W_x^1 &= J_{x,1}^4(W_{z_4}) \\
W_x^2 &= J_{x,2}^4(W_{z_4}) \\
W_x^3 &= J_{x,3}^4(W_{z_4}) \\
W_x^4 &= J_{x,4}^4(W_{z_4}) \\
W_y^1 &= J_{y,1}^4(W_{z_4}) \\
W_y^2 &= J_{y,2}^4(W_{z_4}) \\
W_y^3 &= J_{y,3}^4(W_{z_4}) \\
W_y^4 &= J_{y,4}^4(W_{z_4}) \\
W_z^1 &= J_{z,1}^4(W_{z_4}) \\
W_z^2 &= J_{z,2}^4(W_{z_4}) \\
W_z^3 &= J_{z,3}^4(W_{z_4}) \\
W_z^4 &= J_{z,4}^4(W_{z_4})
\end{aligned}
\tag{A.12}
$$

Where W_{z_4} is the constant corresponding to the location of the wrist on the z-axis of the frame-4 system, and $J_{x,i}^4(W_{z_4})$ is the Jacobian with respect

to the point W_{z_4} on the frame-4 z-axis.

Bibliography

[1] Anderson, T. L. and Donath, M., "Animal behavior as a paradigm for Developing Robot Autonomy," *International journal of robotics and autonomus systems*, 6:145–168, June 1990.

[2] Andrews, J. R. and Hogan, N., "Impedance Control as a Framework for Implementing Obstacle Avoidance in a Manipulator," *Control of Manufacturing Processes and Robotic Systems*, 243–251, 1983.

[3] Avnaim, F., Boissonnat, J., and Faverjon, B., "A Practical Exact Motion Planning Algorithm for Polygonal Objects Amidst Polygonal Obstacles," Technical Report 890, INRIA, Sophia–Antipolis, France, 1988.

[4] Barraquand, J. and Latombe, J., "Robot Motion Planning: A Distributed Representation Approach," Technical Report STAN-CS-89-1257, Department of Computer Science, Stanford University, 1989.

[5] Beckerman, M. and Oblow, E. M., "Treatment of Systematic Errors in the Processing of Wide-Angle Sonar Sensor Data for Robotic Navigation," *IEEE Transactions on Robotics and Automation*, 6(2):137–145, April 1990.

[6] Beer, R. D., Chiel, H. J., and Sterling, L. S., "A biological perspective on Autonomus Agent Design," *International journal of robotics and autonomus systems*, 6:169–186, June 1990.

[7] Beer, R. D., Chiel, H. J., Quinn, R. D., and Larsson, P., "A Distributed Neural Network Architechture for Hexapod Robot Locomotion," in *Neural Computation*, Brooks, R., editor, pages 356–365, Massachusetts Institute of Technology, 1992.

[8] Bekey, G. and Tomovic, R., "Robot Control by Reflex Actions," in *Proceedings of International Conference on Robotics and Automation*, pages 240–247, April 1986.

[9] Branicky, M. S., *Efficient Configuration Space Transforms for Real-Time Robotic Reflexes.* Master's thesis, Case Western Reserve University, Department of Electrical Engineering and Applied Physics, February 1990.

[10] Branicky, M. S., "Stability of Switched and Hybrid Systems," Tech. Report LIDS-P-2214, Lab. for Information and Decision Systems, Massachussetts Instituite of Technology, Boston, MA, November 1993.

[11] Branicky, M. S. and Newman, W. S., "Rapid Computation of Configuration Space Obstacles," in *Proceedings of International Conference on Robotics and Automation*, pages 304–310, May 1990.

[12] Brooks, R., "Elephants Don't Play Chess," *International journal of robotics and autonomus systems*, 6:3–15, June 1990.

[13] Brooks, R., "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, RA–2:14, March 1986.

[14] Brooks, R., "Solving the Find-Path Problem by Good Representation of Free Space," *IEEE Transactions on Systems, Man and Cybernetics*, 13(3):190–197, 1983.

[15] Brooks, R. A., "A Hardware Retargetable Distributed Layered Architechture for Mobile Robot Control," in *Proceedings of International Conference on Robotics and Automation*, pages 106–110, April 1987.

[16] Burroughs, M., "Inhibitory Interactions Between Spiking and Non Spiking Local Intern Neurons in the Locust," *The Journal of Neuroscience*, 7(10):3282–3292, October 1987.

[17] Burroughs, M., "Organization of Receptive Fields of Spiking Local Intern Neurons in the Locust with Input from Hair Afferents," *The Journal of Neurophysiology*, 53(5):1147–1156, May 1985.

[18] Burroughs, M., "The processing of Mechanosensory Information by Spiking local Neurons in the Locust," *The Journal of Neurophysiology*, 54(3):463–477, September 1985.

[19] Burroughs, M., "Proprioceptive Inputs to Nonspiking Local Intern neurons contribute to Local reflexes of the Locust Hindleg," *The Journal of Neuroscience*, 8(8):, August 1988.

[20] Burrows, M. and Laurent, G., *The Computing Neuron*, chapter Reflex Circuits and the Control of Movement, page .

[21] Burrows, M. and Laurent, G., "Reflex Circuits and the Control of Movement," in *The computing neuron*, Durbin, R., Miall, C., and Mitchison, G., editors, chapter 13, pages 244–261, Addison-Wesley Publishing Company, 1989.

[22] Canny, J. *The Complexity of Robot Motion Planning.* The MIT Press, Cambridge, Massachussets, 1988.

[23] Canny, J., "A New Algebraic Method for Robot Motion Planning and Real Geometry," in *Proceedings of International Conference on Robotics and Automation*, pages 808–813, April 1988.

[24] Capek, K., "Rossum's Universal Robot," English version by P. Selver and N. Playfair. New York: Doubleday, Page and Company, 1923.

[25] Chen, P. and Hwang, Y., "SANDROS: A Motion Planner with Performance Proportional to Task Difficulty," in *Proceedings of International Conference on Robotics and Automation*, pages 2346–2353, May 1992.

[26] Chiel, H. J., Beer, R. D., Quinn, R. D., and Espenschied, K. S., "Robustness of a Distributed Neural Network Controller for Locomotion in a Hexapod Robot," *IEEE Transactions on Robotics and Automation*, 8(3):293–303, June 1992.

[27] Donald, B., *Error Detection and Recovery in Robotics.* Springer-Verlag, 1989.

[28] Faverjon, B. and Tournassoud, P., "A Local Approach for Path Planning of Manipulators with a High Number of Degrees of Freedom," in *Proceedings of International Conference on Robotics and Automation*, pages 1152–1159, April 1987.

[29] Flynn, A. and Brooks, R., "MIT Mobile Robots What's Next?," in *Proceedings of International Conference on Robotics and Automation*, pages 611–617, April 1988.

[30] Flynn, A., Brooks, R., Wells, W., and Barret, D., "Intelligence for Miniature Robots," *Sensors and Actuators*, 20:187–196, 1989.

[31] Hahn, W., *Theory and Application of Liapunovs Direct Method*, chapter Generalizations of the Concept of Stability, pages 129–150. Prentice–Hall, Inc., Englewood Cliffs, New Jersey 07632, 1963.

[32] Hogan, N., "Impedance Control: An Approach to Manipulation: Part III–Applicaitions," *Journal of Dynamic Systems Measurement, and Control*, 107:17–24, March 1985.

[33] Hwang, Y. H., "Boundary Equations of Configuration Obstacles for Manipulators," in *Proceedings of International Conference on Robotics and Automation*, pages 298–303, May 1990.

[34] Hwang, Y. and Ahuja, N., "Gross Motion Planning—A Survey," *ACM computing surveys*, 24(3):219–291, September 1992.

[35] Kaelbling, L. and Rosenschein, S., "Action and Planning in Embedded Agents," *International Journal of Robotics and Autonomous Systems*, 6(1–2):35–48, June 1990.

[36] Khatib, O., "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *Journal of Robotics Research*, 5(5):90–98, Spring 1986.

[37] Khatib, O. and Mampey, L. *Fonction decision-commande dùn robot manipulateur*. Rep. 2/7156 DERA/CERT, Toulouse, France, 1978.

[38] Koditschek, D. E., "Exact Robot Navigation by Means of Potential Functions," in *Proceedings of International Conference on Robotics and Automation*, pages 1–6, April 1987.

[39] Krishnaswamy, V. K., *On-Line Motion Planning in three dimensional Configuration Space for a Robotic Manipulator*. Master's thesis, Case Western Reserve University, Department of Electrical Engineering and Applied Physics, October 1990.

[40] Krishnaswamy, V. K. and Newman, W. S., "On-Line Collision Motion Planning Using Critical Point Graphs in Two-dimensional Configuration Space," in *Proceedings of International Conference on Robotics and Automation*, pages 2334–2339, May 1992.

[41] Krogh, B., "A Generalized Potential Field Approach to Obstacle Avoidance Control," in *SME Conf. Proc. Robotics Research: The Next Five Years and Beyond*, page , August 1984.

[42] LaSalle, J. P., "Some Extensions of Liapunov's Second Method.," *IRE Transactions profess.group.circuit theory*, CT-7:520–527, 1960.

[43] Latombe, J., *Robot Motion Planning. A Kluwer Academic Publishers series*, Kluwer Academic Publishers, Boston, 1986.

[44] Lewis, C. L. and Maciejewski, A. A., "Optimization of the Dynamic Performance of Redundant Robots in the Presence of Faults," in *Fourth International Symposium on Robotics and Manufacturing*, pages 279–284, November 1992.

[45] Lozano-Pérez, T., "Automatic Planning of Manipulator Transfer Movements," *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(10):681–698, October 1981.

[46] Lozano-Pérez, T., "Spatial Planning: a Configuration Space Approach," *IEEE Trans. on Computers*, C-32(2):108–120, February 1983.

[47] Lozano-Pérez, T. and Wesley, M., "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *Communications of the ACM*, 22(10):560–570, October 1979.

[48] M. L. Visinsky, J. C. and Walker, I. D., "Expert System Framework for Fault Detection and Fault Tolerance in Robotics," in *Fourth International Symposium on Robotics and Manufacturing*, pages 793–800, November 1992.

[49] Maciejewski, A. A., "The Design and Control of Fault Tolerant Robots for Use in Hazardous or Remote Environments," in *Proceedings of the fourth American Nuclear Society Topological Meeting on Robotics and Remote Systems*, pages 633–642, February 1991.

[50] Maciejewski, A. A., "Fault Tolerant Properties of Kinematically Redundant Manipulators," in *Proceedings of International Conference on Robotics and Automation*, pages 638–642, May 1990.

[51] Maes, P., "Situated Agents Can Have Goals," *International Journal of Robotics and Autonomous Systems*, 6(1–2):49–70, June 1990.

[52] Miyazaki, F. and Arimoto, S., "Sensory Feedback Based on the Artificial Potential for Robots," in *Proceedings of the 9th Triannual World Congress of International Factory Automation*, pages 2381–2386, July 1984.

[53] Moon, F. C., *Chaotic and Fractal Dynamics, an introduction for applied scientists and engineers.* John Wiley and Sons, Inc, New York, NY, 1992.

[54] Newman, W. S., "Automatic Obstacle Avoidance at High Speeds via Reflex Control," in *Proceedings of International Conference on Robotics and Automation*, pages 1104–1109, May 1989.

[55] Newman, W. S., "High Speed Robot Control and Obstacle Avoidance Using Dynamic Potential Functions," in *Proceedings of International Conference on Robotics and Automation*, pages 14–24, April 1987.

[56] Newman, W. S., *High Speed Robot Control in Complex Environments.* PhD thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering, October 1987.

[57] Newman, W. S. and Branicky, M. S., "Experiments in Reflex Control for Industrial Manipulators," Report TR-89-153, Center for Automation and Intelligent Systems Research, Case Western Reserve University, Cleveland, Ohio, October 1989.

[58] Newman, W. S. and Branicky, M. S., "Experiments in Reflex Control for Industrial Manipulators," in *Proceedings of International Conference on Robotics and Automation*, pages 266–271, May 1990.

[59] Newman, W. S. and Branicky, M. S., "Real-Time Configuration-Space Transforms for Obstacle Avoidance," *The International Journal of Robotics Research*, 10(6):650–667, December 1991.

[60] Nilsson, N., "A Mobile Automaton: An Application of Artificial Intelligence Techniques," in *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, pages 509–520, 1969.

[61] Ó'Dúnlaing, C., Sharir, M., and Yap, C., "Retraction: A New Approach to Motion Planning," in *Proceedings of the 15th ACM Symposium on the Theory of Computing*, pages 207–220, 1983.

[62] Paden, B., Mees, A., and Fisher, M., "Path Planning Using a Jacobian—Based Freespace Generation Algorithm," in *Proceedings of International Conference on Robotics and Automation*, pages 1732–1737, May 1989.

[63] Pavlov, V. and Voronin, A., "The Method of Potential Functions for Coding Constraints of the External Space in an Intelligent Mobile Robot," in *Soviet Auto. Cont. 6*, page , 1984.

[64] Payton, D., "An Architechture for Reflexive Autonomous Vehicle Control," in *Proceedings of International Conference on Robotics and Automation*, pages 1838–1845, April 1986.

[65] Rimon, E. and Koditschek, D. E., "The Construction of Analytic Diffeomorphisms for Exact Robot Navigation on Star Worlds," in *Proceedings of International Conference on Robotics and Automation*, pages 21–26, April 1988.

[66] Schwarz, J. and Sharir, M., "On the 'Piano Movers' Problem: I. The Case if a Two-Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers," *Communications on Pure and Applied Mathematics*, 36:345–398, 1983.

[67] Schwarz, J. and Sharir, M., "On the 'Piano Movers' Problem: II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds," *Advances in Applied Mathematics*, 298–351, 1983.

[68] Seraji, H., "Configuration Control of Redundant Manipulators: Theory and Implementaion," *IEEE Transactions on Robotics and Automation*, 5(4):472–490, August 1989.

[69] Slotine, J. E. and Li, W., *Applied Nonlinear Control*, chapter Fundamentals of Lyapunov Theory, pages 41–99. Prentice–Hall, Inc., Englewood Cliffs, New Jersey 07632, 1991.

[70] Tomovic, R., Bekey, G., and Karplus, W., "A Strategy for Grasp Synthesis with Multifingered Robot Hands," in *Proceedings of International Conference on Robotics and Automation*, pages 83–89, April 1987.

[71] Tomovic, R. and Boni, G., "An Adaptive Artificial Hand," *IRA Transactions on Automatic Control*, AC7:3–10, 1962.

[72] Tomovic, R. and Stojilkovic, Z., "Multifunctional Terminal Device with Adaptive Grasping Force," *Automatica*, 11:567, 1975.

[73] Vidyasagar, M. M., *Nonlinear Systems Analysis*. Prentice–Hall, Inc., Englewood Cliffs, New Jersey 07632, 2nd ed edition, 1993.

[74] Wikman, T. S., Branicky, M. S., and Newman, W. S., "Reflex Control for Robot System Preservation, Reliability, and Autonomy," *Electrical Engineering and Computers an International Journal*, Pending():, 1994.

[75] Wikman, T. S., Branicky, M. S., and Newman, W. S., "Reflexive Collision Avoidance: A Generalized Approach," in *Proceedings of International Conference on Robotics and Automation*, pages 31–36, May 1993.

[76] Wikman, T. S. and Newman, W. S., "A Fast, On-Line Collision Avoidance Method for a Kinematically Redundant Manipulator Based on Reflex Control," in *Proceedings of International Conference on Robotics and Automation*, pages 261–267, May 1992.

[77] Wikman, T. S. and Newman, W. S., "Reflex Control for Robot System Preservation and Reliability," in *Proceedings of the 1992 International Symposium on Robotics and Manufacturing*, pages 979–986, Santa Fe, NM, November 1992.

[78] Wikman, T. S. and Newman, W. S., "Ultrasound Based World Mapping Using Reflex Control," Report, Center for Automation and Intelligent Systems Research, Case Western Reserve University, Cleveland, Ohio, May 1994.

[79] Wong, H. C. and Orin, D. E., "Reflex Control of the prototype leg during contact and slippage," in *Proceedings of International Conference on Robotics and Automation*, pages 808–813, April 1988.

[80] Wu, E., Hwang, J., and Chladek, J., "Fault Tolerant Joint Development for the Space Shuttle Remote Manipulator System: Analysis and Experiment," in *Fourth International Symposium on Robotics and Manufacturing*, pages 505–510, November 1992.