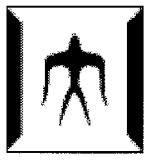
Obstacle avoidance with model predictive control in a hybrid controller

TU/e technische universiteit eindhoven



Tokyo Institute of Technology

Obstacle Avoidance with Model Predictive Control in a Hybrid Controller

J.J.A.M. Keij

DCT 2002.02

Supervisors: professor J.-I. Imura (Tokyo Institute of Technology, Japan)
professor H. Nijmeijer (Eindhoven University of Technology, Holland)
dr.ir. H.A. van Essen (Eindhoven University of Technology, Holland)

Tokyo/Eindhoven, January 2002

Obstacle Avoidance with Model Predictive Control in a Hybrid Controller

Joep Keij

"It is rather hard work: There is no smooth road into the future: But we go round, or scramble over the obstacles."
- D.H. Lawrence, Lady Chatterley's Lover

Traineeship conducted at Tokyo Institute of Technology, Japan (June – September 2001)

TABLE OF CONTENTS

INTRODUCTION

This report is the result of a 14-week traineeship conducted at the Tokyo Institute of Technology, Japan in the summer of 2001. The main goal of the traineeship is to design an obstacle avoidance algorithm for a robot, making use of a hybrid controller with a Model Predictive Controller routine incorporated.
Obstacle avoidance is a quite active region in robotics. Especially in human-robot interactive environments and space robotics some very intelligent solutions have been developed. It is clear that an algorithm for a guidance robot in a museum or a mail-delivering robot in an office environment needs a completely different obstacle avoidance algorithm than a planetary exploring robot in the rocky environment of planet Mars. Nevertheless an algorithm is designed in as general terms as possible to describe a wide area of potential applications. Since practical problems in application, like sampling times and delay, are mainly neglected in this report, it deserves recommendation to be very critical in application of the presented theory.
Obstacle avoidance is a more or less arbitrary example of a possible hybrid controllable application. Other possibilities are e.g. chemical plants with a potentially dangerous situation, which has to be solved (too high temperature, concentrations). The more general goal of this traineeship is to construct a framework of Model Predictive Control for hybrid systems in *emergency mode*. It is quite difficult to obtain an obstacle avoidance algorithm without specific details of the robot and environment. Therefore some assumptions have to be made to concretise the situation and develop a working example of a model predictive controller for hybrid systems. These assumptions are presented in this report where necessary.
In this report a design is presented and analysed. A working example is developed, although a lot of improvements and extensions are needed to apply this method to any practical configuration in the near future. Although this report will not present a complete in depth analysis, some suggestions of improvement will be given at the end of this report.
To design a suitable hybrid controller for the given example a short literature orientation in existing obstacle avoidance methods has been conducted, the results will be presented in chapter 2. In chapter 3 and 4 the principles of respectively model predictive control and hybrid controllers are introduced. The choice of a cost function for the MPC controller is critical for the performance of this example. A brief discussion about this choice is presented in chapter 5. Finally in chapter 6 MPC is applied to the hybrid system of the example and the results are reported and discussed.

CHAPTER 1 - STATEMENT OF THE PROBLEM

The main problem in application of Model Predictive Control in general is the large computational demand. To solve the most common control problems with MPC relatively large calculation times are required, because the matrix formulations imply large matrix structures. As the calculation time in MPC applications is the main limiting factor to small sample times, the performance of the system decreases. This statement holds even for very simple systems but rises for hybrid systems in specific. The optimisation for this class of systems is in general very complex, although some good algorithms are available the calculation times are vast.

Yet MPC is a very powerful control strategy, among others because it can take constraints on the states (outputs) and input signals into account. How can MPC be applied, without losing too much performance but making use of the advantages of this strategy? One possibility is a control strategy with a Model Predictive Controlled *emergency mode*, such as an 'avoidance mode'.

> The goal of this traineeship is *to design a hybrid controller for hybrid systems, making use of a model predictive controller in emergency mode.*

This will be illustrated with the following example; an autonomous mobile robot is controlled to follow a (known) trajectory to some desired point. This task is supposed to be relatively simple and will be carried out with a simple digital (tracking) controller. On its way some obstacle, which blocks the trajectory, is detected. At that point the control strategy is switched to a more *intelligent* Model Predictive Controller, which will try to avoid the obstacle and continue its way to the desired point or trajectory. Because of the more computational demanding MPC strategy, the robot probably has to decrease its speed in exchange for a save trajectory around the obstacle(s).

CHAPTER 2 – OBSTACLE AVOIDANCE

2 GENERAL INTRODUCTION

This chapter gives a brief introduction into the field of obstacle avoidance. To avoid an obstacle can be translated into the problem to find a suitable path or trajectory considering the given environmental constraints. As a starting point for the design process to obtain a new obstacle avoidance algorithm a brief literature exploration has been conducted. This chapter presents some methods and principles found in literature. This literature research can be roughly separated in a general motion planning part and some examples of existing methods to illustrate the general principles. For a more extensive treatment of the presented methods one is referred to the original articles in literature.

2.1 MOTION PLANNING

In general a motion plan consists of the kinematic trajectory for the system as well as the actuator forces that move the system along the trajectory. The actuator forces can be sometimes obtained from the given kinematics in a particularly simple way. In other instances we use the kinematic description of the system because the dynamic equations are difficult to derive. There are also cases when we employ the kinematic model of the system to abstract the details of the actuation scheme. In literature the term *dynamic motion planning* is used when the actuator forces are part of the computed motion plan and *kinematic motion planning* is used when only the kinematic trajectory for the system is computed.

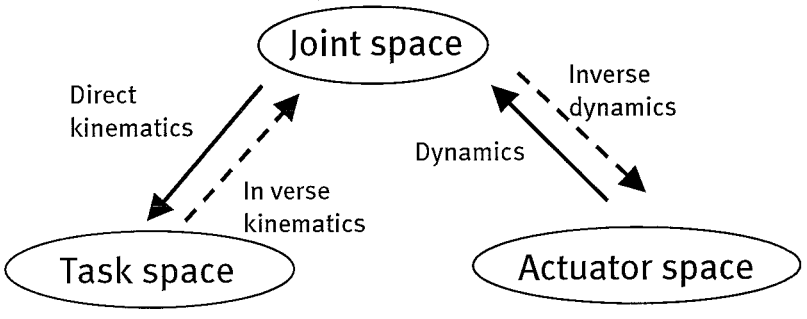


Figure 2.1: *Spaces in which motion can be observed and mappings between*

Motion planning can furthermore be separated into two categories; *explicit motion planning* if the motion plan is computed before the motion is executed, and *implicit motion planning* if the trajectory and the actuator forces are computed while the system moves. Please note that if we want to optimise the performance of the motion or guarantee certain properties of the trajectory in general an explicit motion plan has to be used. If it only matters that a desired configuration is reached, implicit schemes are considered to be sufficient.
The division into explicit and implicit schemes for motion planning also applies to trajectory generation in robotics. In most cases, explicit schemes are used for kinematic motion planning while implicit schemes are usually employed for dynamic motion planning.

2.2 Implicit Methods

Implicit schemes only use the information about the state of the robot and the environment to compute how to move and can be interpreted as feedback mechanisms. They are very attractive from a computational point of view since no processing is required prior to the motion. The simplest scheme, corresponding to the final position control in biological systems, is to make the set-point for the joint controllers equal to the desired final position in the joint space and let the error between the current position and the set-point drive the robot. A modification of this scheme where the velocity during the motion is appropriately shaped is often provided on industrial robots as one of the possible modes of motion, but it is hardly useful for large amplitude motions. One of the reasons is that the shape of the trajectory in the taskspace depends on the location of the start and the end-configuration within the joint space. If obstacles are present in the workspace of a robot, it is difficult to predict whether the robot will avoid them or not.
Another possibility is to define a potential function with the equilibrium point at the goal configuration. The actuators of the robot are programmed to generate the force dictated by the potential field, driving the robot towards the goal configuration. This scheme is much more flexible than the final position control since the potential field that guides the motion can be chosen. It is also very easy to implement obstacle avoidance by assigning a repulsive potential to each obstacle. This method has evolved in a method, in which the range of the repulsive potential is limited, so that only the obstacles that are close to the robot will affect the motion. The robot thus only needs to know local information about the environment. If the potential is defined in the joint space, the problem of kinematic redundancy can be resolved as well. The main drawback of the potential function method is that there may exist local minima that can *trap* the robot. Rimon and Koditschek [Rimon, 1992] demonstrated that a potential (navigation) function can be constructed which has a global minimum and for which all other equilibrium points are saddle-points (unstable equilibria) that lie in a set of measurement zero. However, constructing such a navigation function requires complete knowledge of the space topology and many advantages of the *original potential function method* are lost. Another deficiency of potential fields is that the generated trajectories are usually far from being of minimal length. Finally, it is difficult to take various constraints posed by the task into account such as velocity limits or nonholonomic constraints.

2.3 Explicit Methods

To compute a trajectory, explicit methods (also referred to as open-loop schemes) require knowledge of the global properties of the space. The advantage of such schemes is that task requirements can be taken into account during the planning process. The approach is also attractive from the control point of view: once the trajectory of the system is planned, the system can be linearised along this trajectory and methods from linear control theory can be used to control its motion.
One possible subclass are roadmap methods, which construct a set of curves, called roadmap, that *sufficiently connect* the space. A path between two arbitrary points is found by choosing a curve on the roadmap and connecting each of the two points to this curve with a simple arc. Instead, cell decomposition methods divide the configuration space into non-overlapping cells and construct a connectivity graph expressing the neighbourhood relations between the cells. The cells are chosen so that a path between any two points in the cell is easily found. To find a trajectory between two points in the

configuration space, a *corridor* is first identified by finding a path between two points in the connectivity graph. Subsequently, a path in the configuration space is obtained by appropriately connecting the cells that form the corridor. The most general versions of roadmap and cell decomposition methods work for cases in which obstacles in the configuration space can be described as semi-algebraic sets. However, most practical implementations assume that the obstacles and the robot can be described as polygons. At the price of considerably increased complexity, it is also possible to extend some of the approaches to cases in which the obstacles in the environment move and sensors provide their position.
A common feature of all the motion planning schemes described in this section is that they are based on discrete algorithms. In one way or another the configuration space is discretized and represented by a graph. Subsequently, trajectory planning is reduced to finding a path in this graph. These methods are purely kinematic: they only generate a trajectory in the configuration space, while the dynamics of the robot and the possible constraints on the actuator forces are not taken into account. To obtain a trajectory in the actuator space a separate mechanism must be employed. From the point of view of hierarchical organisation, they therefore assume separate planning at each of the three levels: task space, joint space and actuator space.
[Zefran, 1996] illustrates some more separations as follows in figure 2.2 .

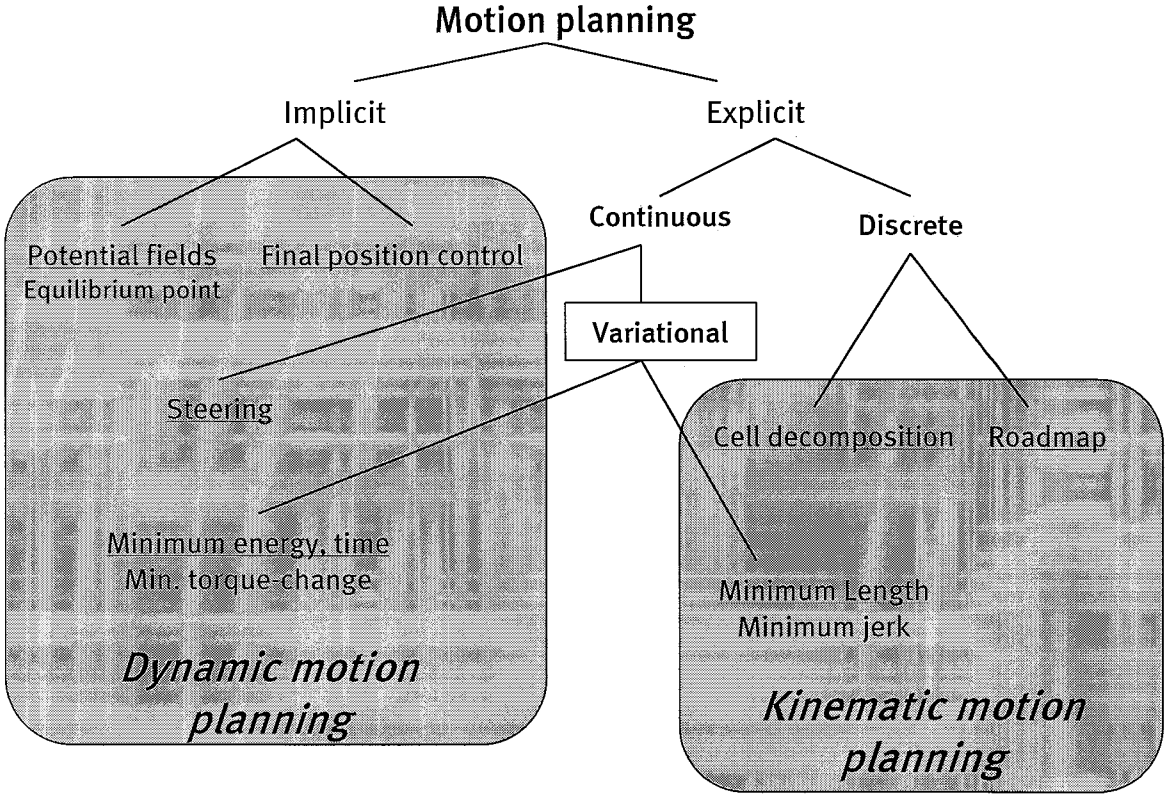


Figure 2.2: *Division of the approaches to motion planning found in literature*

CHAPTER 3 - INTRODUCTION TO MODEL PREDICTIVE CONTROL

Model Predictive Control (MPC) refers to a class of discrete time controllers, which base the input signal on a prediction of future outputs of the system (process). These predictions are based on a model of the system (process) that is to be controlled. The main technique behind this concept is the principle of receding or moving horizons. Due to the model-based approach the online optimisation can take constraints in to account with respect to input signals, controlled and uncontrolled states. Because of the large calculations due to this principle, MPC is most suitable and most applied for relatively 'slow' processes. Although due to fast increasing processor capabilities an increasing number of processes could be controlled with a Model Predictive Controller nowadays.
In the following sections a brief overview of the principles and features of MPC will be given. More information can be found in the lecture notes [Van Essen, 2000]. A more extensive overview is given in [Maciejowski, 2002].

3.1 CONCEPT OF MPC

The scheme presented in figure 2.1 describes the principle of receding horizons that is applied to Model Predictive Controllers. For convenience only one input and one state is considered but for MIMO systems this principle holds just as well.

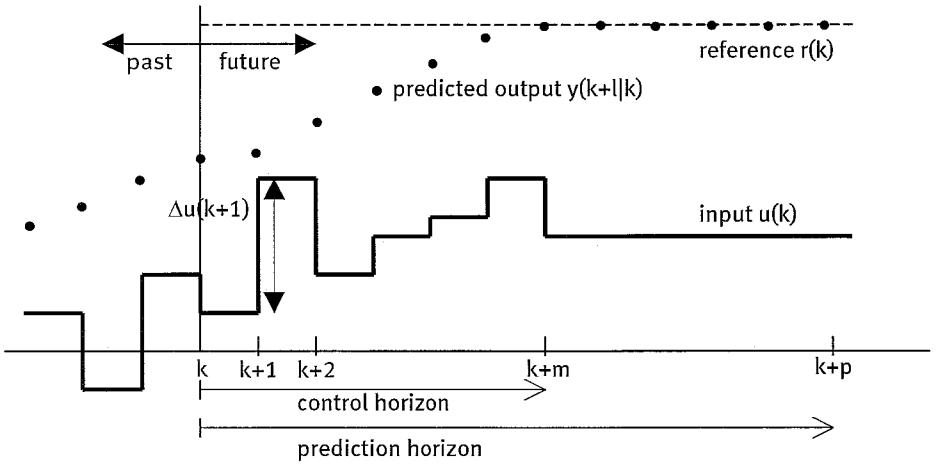


Figure 3.1: *Concept of Model Predictive Control*

At present time k, the response of the output is predicted over a prediction horizon with a length of p samples. The prediction is based on past inputs, current model states (or estimates of the states), latest process measurements, proposed future inputs, and if possible the predicted setpoint disturbances. The manipulated variables are allowed to vary over a control horizon with a length of m samples. The optimal input changes Δu are calculated by minimising a quadratic objective function in the tracking error (y-r) and the input changes Δu . Only the input(change) of the next sample is implemented. The next sample the whole procedure is repeated. This way the horizons are moving in time: receding horizons.
To minimise future deviations of the controlled variables from their reference values (setpoints or trajectories), while preventing the inputs from changing inadmissibly fast, the next (common for MPC) quadratic objective function (in y – r and Δu) is used.

$$\min_{\Delta u(k+1)\ldots\Delta u(k+m)} \sum_{l=2}^{p} [\mathbf{Q}(y(k+l \mid k) - r(k+l))]^2 + \sum_{l=1}^{m} [\mathbf{R}(\Delta u(k+l))]^2 \tag{3.1}$$

Inspecting the quadratic objective function in the (filtered) process output, the reference signal and the input change Δu we see two weighting matrices. **Q** represents the weightings of the setpoints over the prediction horizon. The matrix **R** represents the weighting of the input changes Δu over the control horizon. These weightings may change over the horizons. Furthermore, $y(k+l \mid k)$ denotes the estimate of $y(k+l)$ obtained at sample k, taking into account all (available) information up to and including the current sample k.

3.2 Tuning parameters in MPC

Summarising the tuning parameters from the previous sections, the next parameters are obtained:

- the number of samples in (or the length of) the prediction horizon
- the number of samples in (or the length of) the control horizon
- the sample interval
- the setpoint weighting factors
- the input weighting factors

The controller must be able to observe the consequences of its control actions. Therefore the prediction horizon should exceed the largest time constant of the controlled system, periods of dead-time and periods of inverse response.
System speed decreases when the prediction horizon is increased with respect to the control horizon, although the open-loop robustness of the system increases in this case. A good compromise should be chosen. Basic guidelines for tuning are formulated by [Morari, 1993] on basis of open-loop stable, minimum phase processes, that show responses corresponding to first-order responses.
Increasing lengths of prediction and control horizons, and decreasing the sample interval will increase computation times substantial as the number of d.o.f. in the optimisation rises exponential.
Please not that in this report only one horizon is implemented. The MPC controller is able to influence the controlled variables over the whole prediction horizon.

3.3 Implementation of Linear MPC

In case of a linear optimisation and a non-linear process model an error in introduced. This error can be corrected for by implementing a filter. This filter can be model-based (e.g. (Extended) Kalman filter) or non model-based like the implemented output disturbance filter.

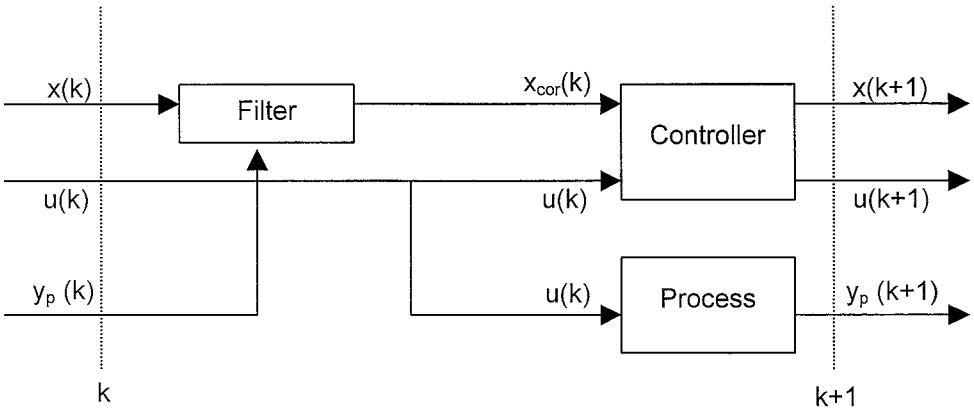


Figure 3.2: *Schematic view of MPC Implementation*

The MPC controller is fed with the current input, the desired setpoint, previous prediction of the states and latest measurements of the process states (outputs). The controller calculates a prediction of the process outputs over the prediction horizon based on the current (unchanged) input and corrected state prediction by means of the (Internal) Prediction Model (IPM). This prediction vector and the current input vector are fed into the Internal Optimisation Model (IOM). The calculation in this optimisation algorithm results in an optimal input change. This input change is summarised with the current input resulting in a new optimal input for the process.

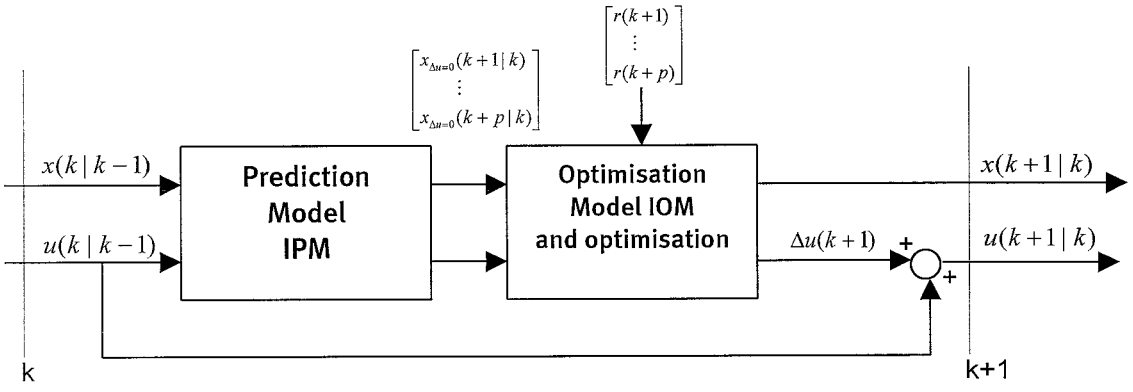


Figure 3.3: *Schematic view of MPC Controller*

Details and principles of the optimisation algorithms can be found in the lecture notes [Van Essen, 2000]. There are roughly two classes of optimisation algorithms. The relatively simple unconstrained optimisation models (e.g. Least Squares LS), and the optimisations that can take constraints into account (e.g. Quadratic Programming QP). Constraints can be formulated with respect to the inputs or the states. Even uncontrolled states can be constrained. These constraints can be upper or lower limits due to safety of physical limitations of the system, or move constraints, like limitations to the actuator.

3.4 NON-LINEAR PREDICTIVE CONTROL

The previous principles all hold for the standard linear Model Predictive Control problem. The unconstrained (Least Squares) optimisation and the constrained optimisation problem (Quadratic Programming) are proven algorithms. Linear approaches will not always be adequate, because most processes behave quite non-linear and can not always be linearised with satisfying results.
Non-linear MPC concepts are developed to deal with these cases. These MPC concepts calculate non-linear predictions over the horizon. Most of these algorithms are computational very demanding because of the increased complexity of the problems. A less demanding solution can be found in successive linearising. In that case the prediction model is linearised around the current setpoint each sample. This extension increases the computational demand, since it requires a sequential (SQP) optimisation problem instead of a single QP problem over the horizon. The main problem with non-linear predictive control is still the possibility of local minima in the optimisation problem.

CHAPTER 4 – INTRODUCTION TO HYBRID SYSTEMS

4.1 INTRODUCTION

It is very important to clearly distinguish between hybrid systems and hybrid controllers. Several definitions are found in literature, therefore the definitions used in this report will be reformulated now. Starting with *hybrid controllers*; this class of controllers is considered to be a multi-controller architecture, which has the capability to *switch* between different control laws or methods during a control task.

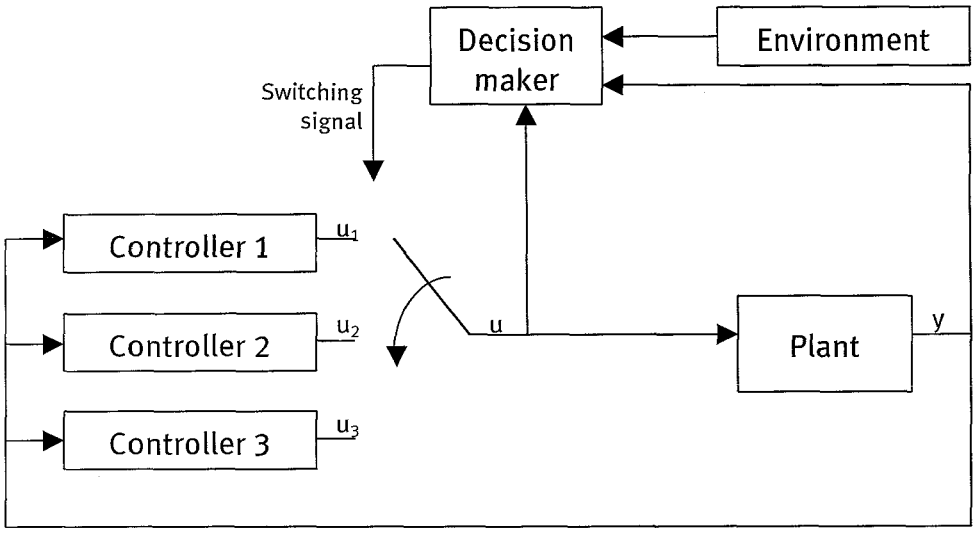


Figure 4.1: *Example of a multi-controller architecture*

In figure 4.1 an example of a multi-controller architecture is illustrated. In this case a hierarchical controlled switch is used to switch from one control law to another, but other methods are also found in literature. A more extensive overview and theoretical fundamentals of switching or hybrid controllers can be found in [Liberzon et al, (1999)].

The second important definition is the definition of *hybrid systems*. Hybrid systems arise in a large area of practical and theoretical applications. In this report hybrid systems are defined as systems which dynamics are described with continuous, logical (binary) and optional with synthesised or auxiliary (binary/continuous) variables. An example of this class of systems are Time-varying Mixed Logical Dynamical (MLD) systems, the general description can be given by:

$$x(t+1) = A_t x(t) + B_{1,t} u(t) + B_{2,t} \delta(t) + B_{3,t} z(t) \quad (4.1a)$$

$$y(t) = C_t x(t) + D_{1,t} u(t) + D_{2,t} \delta(t) + D_{3,t} z(t) \quad (4.1b)$$

$$E_{2,t} \delta(t) + E_{3,t} z(t) \leq E_{1,t} u(t) + E_{4,t} x(t) + E_{5,t} \quad (4.1c)$$

This is a general formulation of a MLD system in linear form, with output y, state x, input u, binary variables δ and auxiliary variables z.

4.2 LOGIC CALCULUS AND LINEAR INTEGER PROGRAMMING

To formulate system-dynamics in terms of logic or binary variables the standard notation, formulated in [Williams (1977); Cavalier et al. (1990); Williams (1993)] is adopted so that capital letters X_i indicate statements, and has a truth value of either "T" (true) or "F" (false). Boolean algebra is used to enable statements to be combined in compound statements by means of connectives; " $\wedge$ " (and), " $\vee$ " (or), " $\sim$ " (not), " $\rightarrow$ " (implies), " $\leftrightarrow$ " (if and only if), " $\oplus$ " (exclusive or). These connectives are defined by means of the truth table presented in (Table 4.1: *Truth table connectives*).

X_1	X_2	$\sim X_1$	$X_1 \vee X_2$	$X_1 \wedge X_2$	$X_1 \rightarrow X_2$	$X_1 \leftrightarrow X_2$	$X_1 \oplus X_2$
F	F	T	F	F	T	T	F
F	T	T	T	F	T	F	T
T	F	F	T	F	F	F	T
T	T	F	T	T	T	T	F

Table 4.1: *Truth table connectives*

This literal X_i notation can be associated with a logical variable $\delta_i \in \{0,1\}$, which has a value of either 1 if X_i = T, or 0 if X_i = F. A set of (compound) statements involving literals $X_1,\ldots,X_n$ can be solved by means of a linear integer program, by translating the statements into linear inequalities involving logical variables δ_i. Some basic transformations, adopted from [Williams (1993)], are stated below in (Table 4.2: *Equivalent statements in literal and binary variables*).

Literal statement		Logical statement
$X_1 \vee X_2$	*is equivalent to*	$\delta_1 + \delta_2 \geq 1$
$X_1 \wedge X_2$	*is equivalent to*	$\delta_1 = 1, \delta_2 = 1$
$\sim X_1$	*is equivalent to*	$\delta_1 = 0$
$X_1 \rightarrow X_2$	*is equivalent to*	$\delta_1 - \delta_2 \leq 0$
$X_1 \leftrightarrow X_2$	*is equivalent to*	$\delta_1 - \delta_2 = 0$
$X_1 \oplus X_2$	*is equivalent to*	$\delta_1 + \delta_2 = 1$

Table 4.2: *Equivalent statements in literal and binary variables*

With this binary toolbox it is quite well possible to model logical parts of processes, like e.g. on/off switches or discrete mechanisms. Especially in case of hybrid systems, which contain both logic as well as continuous dynamics, a link between both worlds is needed. This link can be found in the calculus of *mixed-integer linear inequalities*, i.e. linear inequalities involving both continuous variables $x \in R^n$ and logical variables $\delta \in \{0,1\}$.

The next intermezzo in which such calculus is applied to a literal statement is adapted from [Bemporad, Morari (1998)].

Intermezzo 1:

Consider the statement $X \equiv [f(x) \le 0]$, where $f : R^n \mapsto R$ is linear, assume that $x \in \aleph$, where $\aleph$ is a bounded set, and define

$$M \equiv \max_{x \in \aleph} f(x) \tag{4.2a}$$

$$m \equiv \min_{x \in \aleph} f(x) \tag{4.2b}$$

Theoretically, an over[under]-estimate of M[m] suffices for our purpose. However, more realistic estimates provide computational benefits [Williams (1993), p. 171].
It is easy to verify that

$[f(x) \le 0] \wedge [\delta = 1]$ is true iff $f(x) - \delta \le -1 + m(1-\delta)$, (4.3a)
$[f(x) \le 0] \vee [\delta = 1]$ is true iff $f(x) \le M\delta$ (4.3b)
$\sim [f(x) \le 0]$ is true iff $f(x) \ge \varepsilon$, (4.3b)

where ε is a small tolerance (typically the machine precision), beyond which the constraint is regarded as violated. Therefore

$[f(x) \le 0] \rightarrow [\delta = 1]$ is true iff $f(x) \ge \varepsilon + (m - \varepsilon)\delta$, (4.4a)

$$[f(x) \le 0] \leftrightarrow [\delta = 1] \text{ is true iff } \begin{cases} f(x) \le M(1-\delta) \\ f(x) \ge \varepsilon + (m-\varepsilon)\delta \end{cases} \tag{4.4b}$$

Finally, we report procedures to transform products of logical variables, and of continuous and logical variables, in terms of linear inequalities, which however require the introduction of *auxiliary variables* [Williams (1993), p. 178]. The product term $\delta_1\delta_2$ can be replaced by an auxiliary logical variable $\delta_3 \equiv \delta_1\delta_2$. Then $[\delta_3 = 1] \leftrightarrow [\delta_1 = 1] \wedge [\delta_2 = 1]$, and therefore

$$\delta_3 = \delta_1\delta_2 \text{ is equivalent to } \begin{cases} -\delta_1 + \delta_3 \le 0 \\ -\delta_2 + \delta_3 \le 0 \\ \delta_1 + \delta_2 - \delta_3 \le 1 \end{cases} \tag{4.5}$$

Moreover, the term $\delta f(x)$, where $f : R^n \mapsto R$ and $\delta \in \{0,1\}$, can be replaced by an auxiliary real variable $y \equiv \delta f(x)$, which satisfies $[\delta = 0] \rightarrow [y = 0]$, $[\delta = 1] \rightarrow [y = f(x)]$.

Therefore, by defining *M, m* as in (4.2), $y = \delta f(x)$ is equivalent to:

$$y \le M\delta + \varepsilon, \tag{4.6a}$$

$$y \ge m\delta, \tag{4.6b}$$

$$y \le f(x) - m(1-\delta), \tag{4.6c}$$

$$\varepsilon + y \ge f(x) - M(1-\delta) \tag{4.6d}$$

Alternative methods and formulations for transforming propositional logic problems into equivalent integer programs can be found in the literature, e.g. [Cavalier et al. (1990)].

End of *Intermezzo 1.*

The calculus presented in *Intermezzo 1* will be used to describe linear separations later on in this report. To illustrate the use of this calculus the next example is included. In this example a two-dimensional space is divided in two areas, one area in which some logic variable δ is set to 'True' (or $\delta = 1$) and vice versa. The boundary of the area in this example is set to $-a$.

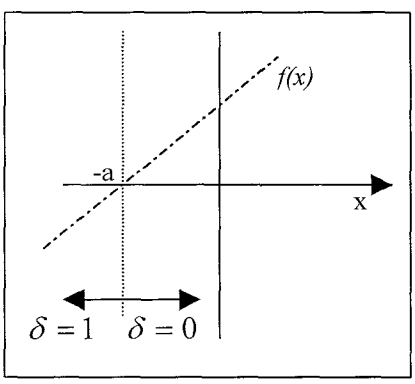


Figure 4.2: *Linear separation*

To obtain this linear separation $f(x)$ is set to $f(x) = x + a$, thus applying the calculus presented in *intermezzo 1*:

$$[x + a \le 0] \leftrightarrow [\delta = 1] \text{ is true iff } \begin{cases} x + a \le M(1-\delta) \\ x + a \ge \varepsilon + (m - \varepsilon)\delta \end{cases} \tag{4.7}$$

Setting $x \in [-10,10]$ we obtain $M = 10 + a$ and $m = -10 + a$.
Checking the several points in the graph it can be seen that this formulation describes the correct linear separation.

CHAPTER 5 – COST FUNCTION AND CONSTRAINTS

The choice of a correct cost function is very important for the performance of the controlled system. A standard cost function is used in predictive controllers. For Mixed Logical Dynamical (MLD) systems usually use some cost function which terms are weighted contributions in terms of the input, binary, auxiliary, continuous state and output variables. This cost function is not the only performance criterion. The formulations of the constraints are at least as important.

5.1 GENERAL STABILISING COST FUNCTION

[Bemporad, (1998)] describes a widely accepted formulation as stated in (5.1) and (5.2), where an equilibrium pair (x_e, u_e) is considered and (δ_e, z_e) are supposed to be definitely admissible.

$$\min_{\{v_0^{T-1}\}} J(v_0^{T-1}, x(t)) \equiv \sum_{k=0}^{T-1} \|v(k)-u_e\|_{Q_1}^2 + \|\delta(k\mid t)-\delta_e\|_{Q_2}^2 + \|z(k\mid t)-z_e\|_{Q_3}^2 + \|x(k\mid t)-x_e\|_{Q_4}^2 + \|y(k\mid t)-y_e\|_{Q_5}^2 \tag{5.1}$$

$$s.t.\begin{cases} x(T|t) = x_e \\ x(k+1|t) = Ax(k|t) + B_1 v(k) + B_2 \delta(k|t) + B_3 z(k|t)\} \\ y(k|t) = Cx(k|t) + D_1 v(k) + D_2 \delta(k|t) + D_3 z(k|t) \\ E_2 \delta(k|t) + E_3 z(k|t) \le E_1 v(k) + E_4 x(k|t) + E_5 \end{cases} \tag{5.2}$$

Where $\|x\|_Q^2 \equiv x'Qx$ and $x(k\mid t) \equiv x(t+k), x(t), v_0^{k-1})$. All weighting matrices are symmetric and positively definite. Under the assumption that the initial state $x(0)$ is such that a feasible solution of the problem given under eq. 5.1 exists at time $t=0$ then $\forall Q_1 = Q'_1 > 0$, $Q_2 = Q'_2 \ge 0$, $Q_3 = Q'_3 \ge 0$, $Q_4 = Q'_4 > 0$ and $Q_5 = Q'_5 \ge 0$ the Mixed Integer Predictive Control (MIPC) law stabilises the system in that

$$\begin{aligned} &\lim_{t\to\infty} x(t) = x_e \\ &\lim_{t\to\infty} u(t) = u_e \\ &\lim_{t\to\infty} \|\delta(t)-\delta_e\|_{Q_2} = 0 \\ &\lim_{t\to\infty} \|z(t)-z_e\|_{Q_3} = 0 \\ &\lim_{t\to\infty} \|y(t)-y_e\|_{Q_5} = 0 \end{aligned} \tag{5.3}$$

while fulfilling the dynamic/relational constraints, as proved in [Bemporad, Morari (1998)] using standard Lyapunov arguments:

Proof 1:

Let U_t^* denote the optimal control sequence $\{v_t^*(0),...,v_t^*(T-1)\}$, let
$V(t) \equiv J(U_t^*, x(t))$
denote the corresponding value attained by the performance index, and let U_1 be the sequence $\{v_t^*(1),...,v_t^*(T-2), u_e\}$. Then U_1 is feasible at time $t+1$, along with the vectors $\delta(k \mid t+1) = \delta(k+1 \mid t)$, $z(k \mid t+1) = z(k+1 \mid t)$, $k = 0,...,T-2$, $\delta(T-1 \mid t+1) = \delta_e$, $z(T-1 \mid t+1) = z_e$, being $x(T-1 \mid t+1) = x(T \mid t) = x_e$ and (δ_e, z_e) definitely admissible. Hence,

$$V(t+1) \le J(U_1, x(t+1))$$
$$= V(t) - \|x(t) - x_e\|_{Q_4} - \|u(t) - u_e\|_{Q_1} - \|\delta(t) - \delta_e\|_{Q_2} - \|z(t) - z_e\|_{Q_3} - \|y(t) - y_e\|_{Q_5}$$

and *V(t)* is decreasing. Since V(t) is lower-bounded by 0, there exists $V_\infty = \lim_{t\to\infty} V(t)$, which implies $V(t+1) - V(t) \to 0$. Therefore, each term of the sum

$$\|x(t) - x_e\|_{Q_4} - \|u(t) - u_e\|_{Q_1} - \|\delta(t) - \delta_e\|_{Q_2} - \|z(t) - z_e\|_{Q_3} - \|y(t) - y_e\|_{Q_5} \le V(t) - V(t+1)$$

converges to zero as well, which proves the theorem.

End of *proof 1*

5.2 SPECIAL CASE COST FUNCTION

In some special cases the general stabilising cost function does not suffice. Especially when the final target can not be reached within the prediction horizon. In the case of an emergency mode hybrid controller, like the one used in this report, this is almost always the case because of the extreme short prediction times, due to the mostly demanded short calculation times in *emergency modes*. In this section these cases will be discussed.
There are several possibilities to adapt the cost function and constraint formulations to the situation where the goal can not be reached within the prediction horizon. This discussion will concentrate on two methods, *decreasing energy method* and the *intermediate target point method.*

5.2.1 DECREASING ENERGY METHOD

The method of decreasing energy is based on the demand that the controlled system has to converge to the desired end point. The cost function is constructed in terms of an energy function. These energy functions contain (quadratic) terms in which the current error between the desired state and the actual state are considered. Besides the desired state also energy terms can be constructed to describe the cost of the input signal, cost to change a binary variable or cost of an auxiliary variable integrated in time.
The controller demands that the total sum of energy functions $V(t)$ decreases every time interval.

Thus the controller demands:
$V(T+1) \leq V(T)$
This method will be able to *guide* the system to the desired state in most cases, but this control strategy does not guarantee that the desired state is reached within reasonable time or even reached ever. One of the possible problem cases is illustrated below.

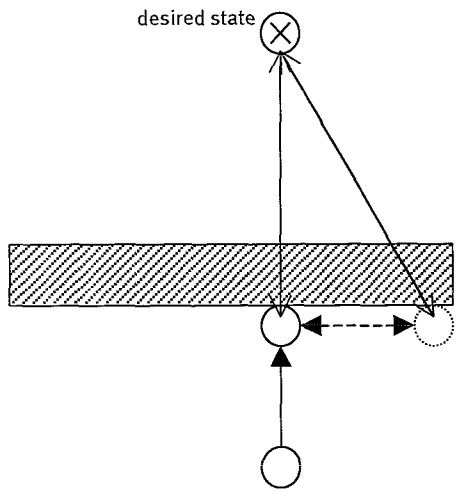


Figure 5.1: *Left-right loop by cost function*

If we consider some cost function based on energy terms in state and input

$$V(t) = \int_0^t (x(\tau) - x_d)^2 + (u(\tau) - u_d)^2 d\tau \qquad (5.4)$$

for a robot encountering an obstacle on its way to some desired point, the robot can not go through the obstacle and will chose to avoid the (infinite) cost of the input signal by going to the right side under the assumption that a short prediction horizon is implemented. At time $t = T+1$ the total energy function is increased by the increase of the distance error $x(t) - x_d$. To meet the demand of declining sum of energies, the robot will now go back to its previous position, and will fall into an infinite loop of going back and forth.

5.2.2 INTERMEDIATE TARGET POINT METHOD

Another method consists of demanding the robot to go to some defined intermediate points and therefore not demanding the robot to be at its final desired state at final time $x(T \mid t) = x_e$, but demanding the robot to be at some intermediate point at some intermediate time $t = T + i$. The main problem is obviously the construction of the intermediate points at intermediate times, but even if these points can be constructed this method does not guarantee the convergence of the system to the desired state. This method is quite sensitive for drifting away from the target if the intermediate points are not defined correctly.

CHAPTER 6 – MODEL PREDICTIVE CONTROL FOR HYBRID SYSTEMS

6.1 EXAMPLE SYSTEM

To illustrate and develop a MPC routine for hybrid systems in case of an emergency mode, like the obstacle avoidance problem, the following test case is designed.

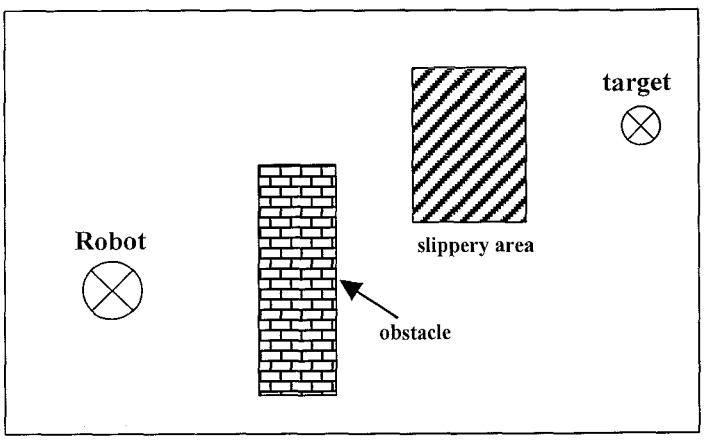


Figure 6.1: *Example configuration*

A (simplified) autonomous mobile robot is to be controlled from its current position to a final (target) position. On its way it will meet an obstacle and an area in which the dynamics of the robot are different (e.g. slippery area, ramp). In this example the robot is supposed to be able to know the complete area between its current position and the target. The dynamics of the robot are supposed to be described by linear equations (or linearised equations) and the system is supposed to be fully controllable. These assumptions are to simplify the problem and to concentrate on the actual avoidance of the obstacle.

6.2 FORMULATION OF THE MPC PROBLEM

The dynamics of the robot are dependent on the position of the robot itself. To describe these changing dynamics a Piecewise Linear Time Invariant dynamic formulation is used.

$$x(t+1) = \begin{cases} A_1 x(t) + B_1 u(t) & if\ \delta_1 = 1 \\ \vdots & \\ A_s x(t) + B_s u(t) & if\ \delta_s = 1 \end{cases} \tag{6.1}$$

Where $\delta_i(t) \in \{0,1\}$, $\forall i = 1,\ldots,s$

(Logic variables satisfying the XOR condition $\bigoplus_{i=1}^{s} [\delta_i(t) = 1]$)

This formulation can not be used in the form of the general formulation of hybrid systems, because it contains products of state, input and logic variables, and therefore it is non-linear.

To avoid a non-linear notation the robot's changing dynamics are reformulated:

$$x(t+1) = \sum_{i=1}^{s} z_i(t) \tag{6.2}$$

With $z_i(t) \equiv [A_i x(t) + B_i u(t)]\delta_i(t)$

and $M = [M_1 \cdots M_n]^T$; $m = [m_1 \cdots m_n]^T$

$$M_j \equiv \max_{i=1,\ldots,s}\left\{\max_{\begin{bmatrix}x\\u\end{bmatrix}\in\ell} A_i^j x + B_i^j u\right\};\ m_j \equiv \min_{i=1,\ldots,s}\left\{\max_{\begin{bmatrix}x\\u\end{bmatrix}\in\ell} A_i^j x + B_i^j u\right\}$$

Bemporad states that (6.2) is equivalent to:

$$\begin{cases} z_i \le M\ \delta_i(t) \\ z_i \ge m\ \delta_i(t) \\ z_i \le A_i x(t) + B_i u(t) - m(1-\delta_i(t)) \\ z_i \ge A_i x(t) + B_i u(t) - M(1-\delta_i(t)) \end{cases} \qquad (6.3)$$

$\forall i = 1,\ldots,s$

To introduce a correct labelling system for the areas of different dynamics a procedure is developed. At first sight this procedure might seem to be much more complicated than strictly necessary. At the start of the project a much more simple system was used to label a area. This system used only one logical variable, if this robotic system is positioned inside area *S* the logical variable of this area (e.g. δ_S) should switch to True (or $\delta_S = 1$). This is illustrated with the next example:

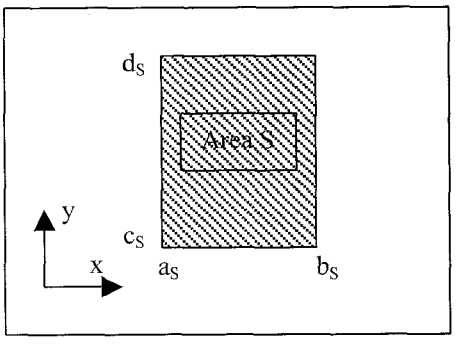


Figure 6.2: *Parametric labelling of areas*

$\delta_S = 1$ if and only if:

$$\begin{cases} a_S \le x \le b_S \\ c_S \le y \le d_S \end{cases}$$

this definition is equivalent to:

$$\begin{cases} x \ge a_S \ \leftrightarrow\ \delta_S = 1 \\ x \le b_S \ \leftrightarrow\ \delta_S = 1 \\ y \ge c_S \ \leftrightarrow\ \delta_S = 1 \\ y \le d_S \ \leftrightarrow\ \delta_S = 1 \end{cases}$$

This concept is used to label the areas in this project, but with some adaptations. If this procedure is extended to multiple areas, this will introduce a problem. Even if the above example is analysed with the extension that outside area *S* different dynamic laws are valid, it will prove very difficult to describe the area outside of area *S*. In this concept of labelling it is not possible to implement a relatively simple 'else...' rule. Therefore a new

concept has been designed to describe all the areas as general as possible but with exclusive labelling of areas. Therefore some intermediate variables are introduced as follows from the next illustration.

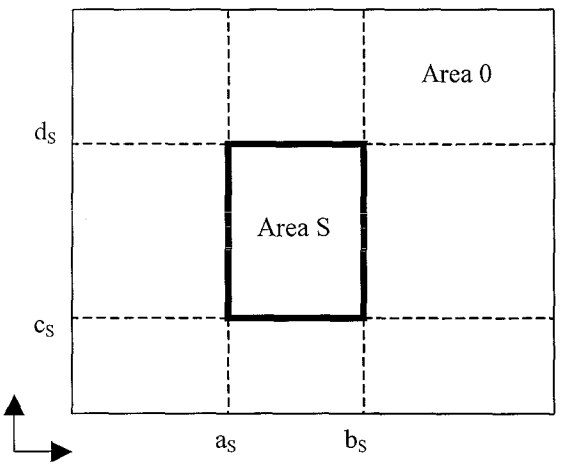


Figure 6.3: *Exclusive labelling of areas*

In figure 6.3 the labelling procedure is carried out as follows. First the area $x \geq a_S$ is labelled with the variable δ_a. This variable is true if and only if the robot is right of the line $x = a_S$. Then the area $x \leq b_S$ is labelled with δ_b, which is true if the robot is left of the line $x = b_S$. The same procedure is applied to the areas above $y = c_S$ and under $y = d_S$, with respectively δ_c and δ_d.

After the labelling of previous areas, two more intermediate binary variables are introduced. If the robot is between the lines $x = a_S$ and $x = b_S$, the variable δ_x is set true, and corresponding: if the robot is between the lines $y = c_S$ and $y = d_S$ the variable δ_y is set true. In the final step of this labelling procedure the binary variable δ_S is introduced. This variable is true if and only if both δ_x and δ_y are true. This procedure, although very extended (7 binary variables), has proved to be able to label the area *S* with exclusion of the other areas.

According to the logic calculus proposed by Bemporad equation (6.3) is equivalent to:

$$\begin{cases} x - a_i \le -\varepsilon - (m-\varepsilon)\delta_i & (6.6a) \\ -x + a_i \le M(1-\delta_i) & (6.6b) \\ x - b_i \le M(1-\delta_i) & (6.6c) \\ -x + b_i \le -\varepsilon - (m-\varepsilon)\delta_i & (6.6d) \\ y - c_i \le -\varepsilon - (m-\varepsilon)\delta_i & (6.6e) \\ -y + c_i \le M(1-\delta_i) & (6.6f) \\ y - d_i \le M(1-\delta_i) & (6.6g) \\ -y + d_i \le -\varepsilon - (m-\varepsilon)\delta_i & (6.6h) \end{cases}$$

These rules are valid for $i = 2 \cdots s$, else $\delta_1 = 1$ (the robot is not in a special area).

6.3 Piecewise Linear Time Invariant dynamics for the example

In the case of the presented example where a robot with a 2-dimensional state space ($n = 2$) moves to a target in an environment with 2 *different areas* (one obstacle and a slippery area) the total number of areas becomes $i = 3$.

Therefor the systems dynamics now written as: (6.7)

$$x(t+1) = z_1 + z_2 + z_3 = B_3 z(t)$$

with: $z(t) = [z_1 \quad z_2 \quad z_3]^T$ and $B_3 = [1 \quad 1 \quad 1]$. (6.8a)

$z_1 = [A_1 x(t) + B_1 u(t)]\delta_1 \Rightarrow$ *normal situation* (6.8b)

$z_2 = [A_2 x(t) + B_2 u(t)]\delta_2 \Rightarrow$ *obstacle* ($A_2 = I; B_2 = 0$) (6.8c)

$z_3 = [A_3 x(t) + B_3 u(t)]\delta_3 \Rightarrow$ *slippery, wet, ramp, etc.*

and e.g. $A_3 = A_1$; $B_3 = 0.1\, B_1$ (*slippery*)

The output is defined as (6.9)

$$y(t) = C\, x(t) = I_2 x(t)$$

And the states and inputs are bounded as described in (6.10) and (6.11) (6.10)

$$x(t) \in [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \quad (6.11)$$

$$u(t) \in [u_{\min}, u_{\max}]$$

The maxima and minima are

$$M = [M_1 \quad M_2]^T \quad (:= [M_x \quad M_y]) \quad (6.12a)$$

$$m = [m_1 \quad m_2]^T \quad (:= [m_x \quad m_y]) \quad (6.12b)$$

There are three kinds of inequality constraints, the first category comes from the labelling of the areas, the second from the definition of the auxiliary variables and the last kind from the restrictions on the input signal.

This results in the following inequality program:

$$\begin{cases} (m-\varepsilon)\delta_i \le -x + (a_i - \varepsilon) & (6.13a) \\ M\delta_i \le x + (M - a_i) & (6.13b) \\ (m-\varepsilon)\delta_i \le x + (-b_i - \varepsilon) & (6.13c) \\ M\delta_i \le -x + (M + b_i) & (6.13d) \\ (m-\varepsilon)\delta_i \le -y + (c_i - \varepsilon) & (6.13e) \\ M\delta_i \le y + (M - c_i) & (6.13f) \\ (m-\varepsilon)\delta_i \le y + (d_i - \varepsilon) & (6.13g) \\ M\delta_i \le -y + (M + d_i) & (6.13h) \end{cases} \quad \text{Labelling}$$

$$\begin{cases} -M\delta_i \le -z_i & (6.13i) \\ -m\delta_i \le z_i & (6.13j) \\ -m\delta_i \le A_i x(t) + B_i u(t) - z_i - m & (6.13k) \\ M\delta_i \le -A_i x(t) - B_i u(t) + z_i + M & (6.13l) \end{cases} \quad \text{Auxiliary variables}$$

$$\begin{cases} -u \le -u_{\min} & (6.13m) \\ u \le u_{\max} & (6.13n) \end{cases} \quad \text{Input bounds}$$

This brings the system description into the convenient form:

$$x(t+1) = Ax(t) + B_1 u(t) + B_2 \delta(t) + B_3 z(t), \quad (6.14a)$$

$$E_2 \delta(t) + E_3 z(t) \le E_1 u(t) + E_4 x(t) + E_5 \quad (6.14b)$$

In which $A = B_1 = B_2 = 0$ and $B_3 = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$. Please note that these values are only valid in these equations, the auxiliary variables all contain a state and input matrix called A_i and B_i. The matrices E_1, E_2, E_3, E_4 and E_5 are presented in appendix A.

To obtain the MPC controller of this MLD system the following cost function J is considered:

$$J = \sum_{t=0}^{T-1} \left\| u(t) - u_f \right\|_{Q_2}^2 + \left\| \delta(t) - \delta_f \right\|_{Q_3}^2 + \left\| z(t) - z_f \right\|_{Q_4}^2 + \left\| x(t) - x_f \right\|_{Q_1}^2 \quad (6.15)$$

subject to:

$$\begin{cases} x(T) = x_f \\ x(t+1) = Ax(t) + B_1 u(t) + B_2 \delta(t) + B_3 z(t) \\ -E_4 x(t) - E_1 u(t) + E_2 \delta(t) + E_3 z(t) \le E_5 \end{cases} \quad (6.16)$$

The elements of this cost function are developed piece-wise to get to the form of (in-) equalities that the Mixed Integer Quadratic Programming (MIQP) solver can handle.

$$\begin{aligned}\sum_{t=0}^{T-1}\left\|u(t)-u_f\right\|_{Q_2}^2 &= \sum_{t=0}^{T-1}\left(u(t)-u_f\right)^T Q_2\left(u(t)-u_f\right)\\ &= \sum_{t=0}^{T-1}\left[u(t)^T Q_2 u(t)-2u_f^T Q_2 u(t)+u_f^T Q_2 u_f\right]\\ &= \begin{bmatrix} u(0)\\ \vdots\\ u(T-1)\end{bmatrix}^T\begin{bmatrix} Q_2 & & \\ & \ddots & \\ & & Q_2\end{bmatrix}\begin{bmatrix} u(0)\\ \vdots\\ u(T-1)\end{bmatrix}-2u_f^T Q_2\begin{bmatrix} I_{mu} & \cdots & I_{mu}\end{bmatrix}\begin{bmatrix} u(0)\\ \vdots\\ u(T-1)\end{bmatrix}+Tu_f^T Q_2 u_f\\ &\equiv \Omega^T\mathbf{Q}_2\Omega-2u_f^2 Q_2\beta_2\Omega+\left(Tu_f^T Q_2 u_f\right)\end{aligned} \tag{6.17}$$

$$\begin{aligned}\sum_{t=0}^{T-1}\left\|\delta(t)-\delta_f\right\|_{Q_3}^2 &= \sum_{t=0}^{T-1}\left(\delta(t)-\delta_f\right)^T Q_3\left(\delta(t)-\delta_f\right)\\ &= \sum_{t=0}^{T-1}\left[\delta(t)^T Q_3(t)-2\delta_f^T Q_3\delta(t)+\delta_f^T Q_3\delta_f\right]\\ &= \begin{bmatrix} \delta(0)\\ \vdots\\ \delta(T-1)\end{bmatrix}^T\begin{bmatrix} Q_3 & & \\ & \ddots & \\ & & Q_3\end{bmatrix}\begin{bmatrix} \delta(0)\\ \vdots\\ \delta(T-1)\end{bmatrix}-2\delta_f^T Q_3\begin{bmatrix} I_{mu} & \cdots & I_{mu}\end{bmatrix}\begin{bmatrix} \delta(0)\\ \vdots\\ \delta(T-1)\end{bmatrix}+T\delta_f^T Q_3\delta_f\\ &\equiv \Delta^T\mathbf{Q}_3\Delta-2\delta_f^2 Q_3\beta_3\Delta+\left(T\delta_f^T Q_3\delta_f\right)\end{aligned} \tag{6.18}$$

$$\begin{aligned}\sum_{t=0}^{T-1}\left\|z(t)-z_f\right\|_{Q_4}^2 &= \sum_{t=0}^{T-1}\left(z(t)-z_f\right)^T Q_4\left(z(t)-z_f\right)\\ &= \sum_{t=0}^{T-1}\left[z(t)^T Q_4(t)-2z_f^T Q_4 z(t)+z_f^T Q_4 z_f\right]\\ &= \begin{bmatrix} z(0)\\ \vdots\\ z(T-1)\end{bmatrix}^T\begin{bmatrix} Q_4 & & \\ & \ddots & \\ & & Q_4\end{bmatrix}\begin{bmatrix} z(0)\\ \vdots\\ z(T-1)\end{bmatrix}-2z_f^T Q_4\begin{bmatrix} I_{mu} & \cdots & I_{mu}\end{bmatrix}\begin{bmatrix} z(0)\\ \vdots\\ z(T-1)\end{bmatrix}+Tz_f^T Q_4 z_f\\ &\equiv \Theta^T\mathbf{Q}_4\Theta-2z_f^2 Q_4\beta_3\Theta+\left(Tz_f^T Q_4 z_f\right)\end{aligned} \tag{6.19}$$

$$\begin{aligned}x(t) &= A^t x(0)+\sum_{i=0}^{t-1}A^i\left[B_1u(t-1-i)+B_2\delta(t-1-i)+B_3z(t-1-i)\right]\\ &= A^t x(0)+\left[A^{t-1},\ldots,A^2,A,I\right]\begin{bmatrix} B_1 & & \\ & \ddots & \\ & & B_1\end{bmatrix}\begin{bmatrix} u(0)\\ \vdots\\ u(T-1)\end{bmatrix}\\ &+\left[A^{t-1},\ldots,A^2,A,I\right]\begin{bmatrix} B_2 & & \\ & \ddots & \\ & & B_2\end{bmatrix}\begin{bmatrix} \delta(0)\\ \vdots\\ \delta(T-1)\end{bmatrix}\\ &+\left[A^{t-1},\ldots,A^2,A,I\right]\begin{bmatrix} B_3 & & \\ & \ddots & \\ & & B_3\end{bmatrix}\begin{bmatrix} z(0)\\ \vdots\\ z(T-1)\end{bmatrix}\end{aligned} \tag{6.20}$$

$$\begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ \vdots \\ x(T-1) \end{bmatrix} = \begin{bmatrix} I \\ A \\ A^2 \\ \vdots \\ A^{T-1} \end{bmatrix} x(0) \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ I & 0 & 0 & & & 0 \\ A & I & 0 & & & \\ \vdots & & & & & \vdots \\ A^{T-2} & A^{T-3} & A^{T-4} & \cdots & I & 0 \end{bmatrix} \begin{bmatrix} B_1 & & & & \\ & B_1 & & & \\ & & B_1 & & \\ & & & \ddots & \\ & & & & B_1 \end{bmatrix} \begin{bmatrix} u(0) \\ u(1) \\ \vdots \\ \vdots \\ u(T-1) \end{bmatrix}$$
$$+ \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ I & 0 & 0 & & & 0 \\ A & I & 0 & & & \\ \vdots & & & & & \vdots \\ A^{T-2} & A^{T-3} & A^{T-4} & \cdots & I & 0 \end{bmatrix} \begin{bmatrix} B_2 & & & & \\ & B_2 & & & \\ & & B_2 & & \\ & & & \ddots & \\ & & & & B_2 \end{bmatrix} \begin{bmatrix} \delta(0) \\ \delta(1) \\ \vdots \\ \vdots \\ \delta(T-1) \end{bmatrix}$$
$$+ \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ I & 0 & 0 & & & 0 \\ A & I & 0 & & & \\ \vdots & & & & & \vdots \\ A^{T-2} & A^{T-3} & A^{T-4} & \cdots & I & 0 \end{bmatrix} \begin{bmatrix} B_3 & & & & \\ & B_3 & & & \\ & & B_3 & & \\ & & & \ddots & \\ & & & & B_3 \end{bmatrix} \begin{bmatrix} z(0) \\ z(1) \\ \vdots \\ \vdots \\ z(T-1) \end{bmatrix} \tag{6.21}$$

$$\begin{bmatrix} x(0) \\ \vdots \\ x(T-1) \end{bmatrix} = G_0 x(0) + G_1 \Omega + G_2 \Delta + G_3 \Theta = G_0 x(0) + [G_1 \quad G_2 \quad G_3] \begin{bmatrix} \Omega \\ \Delta \\ \Theta \end{bmatrix} = G_0 x(0) + G_4 \mathbf{R} \tag{6.22}$$

So that we can write:

$$\begin{aligned} \sum_{t=0}^{T-1} \left\| x(t) - x_f \right\|_{Q_1}^2 &= \begin{bmatrix} x(0) \\ \vdots \\ x(T-1) \end{bmatrix}^T \begin{bmatrix} Q_1 & & \\ & \ddots & \\ & & Q_1 \end{bmatrix} \begin{bmatrix} x(0) \\ \vdots \\ x(T-1) \end{bmatrix} - 2x_f^T Q_1 [I \quad \cdots \quad I] \begin{bmatrix} x(0) \\ \vdots \\ x(T-1) \end{bmatrix} + T x_f^T Q_1 x_f \\ &= \begin{bmatrix} x(0) \\ \vdots \\ x(T-1) \end{bmatrix}^T \mathbf{Q}_1 \begin{bmatrix} x(0) \\ \vdots \\ x(T-1) \end{bmatrix} - 2x_f^T Q_1 \beta_1 \begin{bmatrix} x(0) \\ \vdots \\ x(T-1) \end{bmatrix} + T x_f^T Q_1 x_f \\ &= \left(x(0)^T G_0^T + \mathbf{R}^T G_4^T \right) \mathbf{Q}_1 \left(G_0 x(0) + G_4 \mathbf{R} \right) - 2x_f^T Q_1 \beta_1 \left(G_0 x(0) + G_4 \mathbf{R} \right) \\ &= \mathbf{R}^T G_4^T \mathbf{Q}_1 G_4 \mathbf{R} + 2\left(x_0^T G_0^T \mathbf{Q}_1 G_4 - x_f^T Q_1 \beta_1 G_4 \right) \mathbf{R} + const. \end{aligned} \tag{6.23}$$

The cost function can now be expressed as:

$$\begin{aligned}\tilde{J} &= \Omega^T \mathbf{Q}_2 \Omega - 2u_f^T Q_2 \beta_2 \Omega \\ &+ \Delta^T \mathbf{Q}_3 \Delta - 2\delta_f^T Q_3 \beta_3 \Delta \\ &+ \Theta^T \mathbf{Q}_4 \Theta - 2z_f^T Q_4 \beta_4 \Theta \\ &+ \mathbf{R}^T G_4^T \mathbf{Q}_4 G_4 \mathbf{R} + 2\left(x_0^T G_0^T \mathbf{Q}_1 G_4 - x_f^T \mathbf{Q}_1 \beta_1 G_4\right)\mathbf{R}\end{aligned} \tag{6.24a}$$

$$= \begin{bmatrix}\Omega \\ \Delta \\ \Theta\end{bmatrix}^T \underbrace{\begin{bmatrix}\mathbf{Q}_2 & & \\ & \mathbf{Q}_3 & \\ & & \mathbf{Q}_4\end{bmatrix}}_{\mathbf{Q}} \begin{bmatrix}\Omega \\ \Delta \\ \Theta\end{bmatrix} - 2\underbrace{\begin{bmatrix}u_f^T Q_2 \beta_2 & \delta_f^T Q_3 \beta_3 & z_f^T Q_4 \beta_4\end{bmatrix}}_{\mathbf{P}} \begin{bmatrix}\Omega \\ \Delta \\ \Theta\end{bmatrix}$$

$$+ \mathbf{R}^T G_4^T \mathbf{Q}_4 G_4 \mathbf{R} + 2\left(x_0^T G_0^T \mathbf{Q}_1 G_4 - x_f^T \mathbf{Q}_1 \beta_1 G_4\right)\mathbf{R} \tag{6.24b}$$

$$= \mathbf{R}^T \left[\mathbf{Q} + G_4^T \mathbf{Q}_1 G_4\right]\mathbf{R} + 2\left[\left\{-\mathbf{P} - x_f^T Q_1 \beta_1 G_4\right\} + x_0^T G_0^T \mathbf{Q}_1 G_4\right]\mathbf{R} \tag{6.24c}$$

$$= \mathbf{R}^T S_1 \mathbf{R} + 2\left[S_2 + x_0^T S_3\right]\mathbf{R} \tag{6.24d}$$

The following set of inequalities can be derived with respect to the constraints:

$$\begin{cases} E_2\delta(0) + E_3 z(0) \le E_1 u(0) + E_4 x(0) + E_5 \\ \vdots \\ E_2\delta(T-1) + E_3 z(T-1) \le E_1 u(T-1) + E_4 x(T-1) + E_5 \end{cases} \tag{6.25}$$

In other notation:

$$\begin{bmatrix}E_2 & & \\ & \ddots & \\ & & E_2\end{bmatrix}\begin{bmatrix}\delta(0) \\ \vdots \\ \delta(T-1)\end{bmatrix} + \begin{bmatrix}E_3 & & \\ & \ddots & \\ & & E_3\end{bmatrix}\begin{bmatrix}z(0) \\ \vdots \\ z(T-1)\end{bmatrix} \le \begin{bmatrix}E_1 & & \\ & \ddots & \\ & & E_1\end{bmatrix}\begin{bmatrix}u(0) \\ \vdots \\ u(T-1)\end{bmatrix} + \begin{bmatrix}E_4 & & \\ & \ddots & \\ & & E_4\end{bmatrix}\begin{bmatrix}x(0) \\ \vdots \\ x(T-1)\end{bmatrix} + \begin{bmatrix}E_5 \\ \vdots \\ E_5\end{bmatrix}$$

$$= \mathbf{E}_2\Delta + \mathbf{E}_3\Theta \le \mathbf{E}_1\Omega + \mathbf{E}_4(G_0 x_0 + G_4\mathbf{R}) + \mathbf{E}_5 \tag{6.26a}$$

$$= \begin{bmatrix}-\mathbf{E}_1 & \mathbf{E}_2 & \mathbf{E}_3\end{bmatrix}\begin{bmatrix}\Omega \\ \Delta \\ \Theta\end{bmatrix} - \mathbf{E}_4 G_4 \begin{bmatrix}\Omega \\ \Delta \\ \Theta\end{bmatrix} \le \mathbf{E}_4 G_0 x_0 + \mathbf{E}_5 \tag{6.26b}$$

$$= \mathbf{E}\,\mathbf{R} - \mathbf{E}_4 G_4 \mathbf{R} \le \mathbf{E}_4 G_0 x_0 + \mathbf{E}_5 \tag{6.26c}$$

$$= (\mathbf{E} - \mathbf{E}_4 G_4)\mathbf{R} \le \mathbf{E}_5 + \mathbf{E}_4 G_0 x_0 \tag{6.26d}$$

$$= F_1\,\mathbf{R} \le F_2 + F_3\,x_0 \tag{6.26e}$$

The demand $x(T) = x_f$ can be rewritten as:

$$\begin{aligned}x(T) = A^T x_0 &+ \begin{bmatrix}A^{T-1} & \cdots & A & I\end{bmatrix}\begin{bmatrix}B_1 & & \\ & B_1 & \\ & & B_1\end{bmatrix}\begin{bmatrix}u(0) \\ \vdots \\ u(T-1)\end{bmatrix} \\ &+ \begin{bmatrix}A^{T-1} & \cdots & A & I\end{bmatrix}\begin{bmatrix}B_2 & & \\ & B_2 & \\ & & B_2\end{bmatrix}\begin{bmatrix}\delta(0) \\ \vdots \\ \delta(T-1)\end{bmatrix} \\ &+ \begin{bmatrix}A^{T-1} & \cdots & A & I\end{bmatrix}\begin{bmatrix}B_3 & & \\ & B_3 & \\ & & B_3\end{bmatrix}\begin{bmatrix}z(0) \\ \vdots \\ z(T-1)\end{bmatrix}\end{aligned} \tag{6.27}$$

$$= A^T x(0) + \mathbf{A}\begin{bmatrix}\mathbf{B}_1 & \mathbf{B}_2 & \mathbf{B}_3\end{bmatrix}\mathbf{R} = x_f \tag{6.28}$$

Rewritten in convenient form:

$$\mathbf{A}\begin{bmatrix}\mathbf{B}_1 & \mathbf{B}_2 & \mathbf{B}_3\end{bmatrix}\mathbf{R} = x_f - A^T x(0) \tag{6.29}$$

$$\mathbf{A}_{eq}\,\mathbf{R} = \mathbf{b}_{eq} \tag{6.30}$$

The initial stated MLD system and cost function is now rewritten into:

$$\min_{\mathbf{R}} \mathbf{R}^* S_1 \mathbf{R} + 2(S_2 + x_0^* S_3)\mathbf{R} \tag{6.31}$$

s.t.:
$$F_1\mathbf{R} \le F_2 + F_3 x_0$$
$$A^T x_0 + \mathbf{A}\,\mathbf{B}\,\mathbf{R} = x_f$$

with $\mathbf{R} = \begin{bmatrix}\Omega \\ \Delta \\ \Theta\end{bmatrix} = \begin{bmatrix} u(0) \\ \vdots \\ u(T-1) \\ \delta(0) \\ \vdots \\ \delta(T-1) \\ z(0) \\ \vdots \\ z(T-1)\end{bmatrix}$

The currently used MIQP solver (Matlab routine, miqp.m [Bemporad]) solves problems of the following general form:

$$\min_x x^T(\tfrac{1}{2}H)x + f^T x \tag{6.32}$$

s.t.:
$$\begin{cases} Ax \le b \\ A_{eq}x = b_{eq} \\ v_{lb} \le x \le v_{ub} \\ x \in R^{n_c} \times \{0,1\}^{n_d} \\ x(i_{\text{var}\,type}) \in \{0,1\}^{n_d} \end{cases}$$

This results in the following input arguments for the solver:

$$H = 2\,S_1 \tag{6.33}$$

$$f = 2\,(S_2 + x_0^* S_3)^* \tag{6.34}$$

$$A = F_1 \tag{6.35}$$

$$b = F_2 + F_3 x_0 \tag{6.36}$$

$$A_{eq} = \mathbf{A}\,\mathbf{B} \tag{6.37}$$

$$b_{eq} = x_f - A^T x_0 \tag{6.38}$$

$$\text{var}type = [m_u T + 1, m_u T + 2, \ldots, m_u T + m_\delta T] \tag{6.39}$$

$$lb = [-p \;\cdots\; -p \;\; 0 \;\cdots\; 0 \;\; -p \;\cdots\; -p] \tag{6.40}$$

$$ub = [p \;\cdots\; p \;\; 1 \;\cdots\; 1 \;\; p \;\cdots\; p] \tag{6.41}$$

$with\ p = 1e10$ (or any arbitrary finite large number)

Problems that still have to be addressed are mostly on the field of constructing a correct cost function for the problem with very small prediction times, and designing a convenient criterion for the desired state at each point in time. The MPC controller with very small prediction times cannot reach the target within the chosen very short prediction time. The strategy on what to do in case the target is not reached within the prediction time is not always arbitrary. It is very easy to construct a less restrictive end-point criterion, but the global goal must of course always be the main concern and therefore the convergence of the system must be guaranteed. Weakening the end-state criterion can bring some relieve but implies that the system is not always guaranteed to be convergent. Another possibility is the method of intermediate 'target points'. This means that some algorithm should calculate useful intermediate points, which are reachable within the prediction horizon. This means not only that these points have to be positioned in an obstacle free subspace but also within driving range of the robot. There are a lot of possible combinations of cost functions and end-state criterions. Each possibility reviewed in this traineeship has some positive implications for one area/environment and some negative implications for other environments.
The choice of describing an obstacle as an area with 'infinite difficult' dynamics implies that the cost function must at least contain a term based on input cost. If the cost function is only based on the position error, the obstacle will not be avoided. If it proves to be necessary to construct a cost function solely based on position error, the method must be adapted. In such cases the area can be modelled as a position constrained within the MPC algorithm in stead of an area with difficult dynamics.

6.4 MATLAB IMPLEMENTATION

To test the described method of obstacle avoidance, simulations have been conducted. The simulated situation is illustrated in figure 6.4.

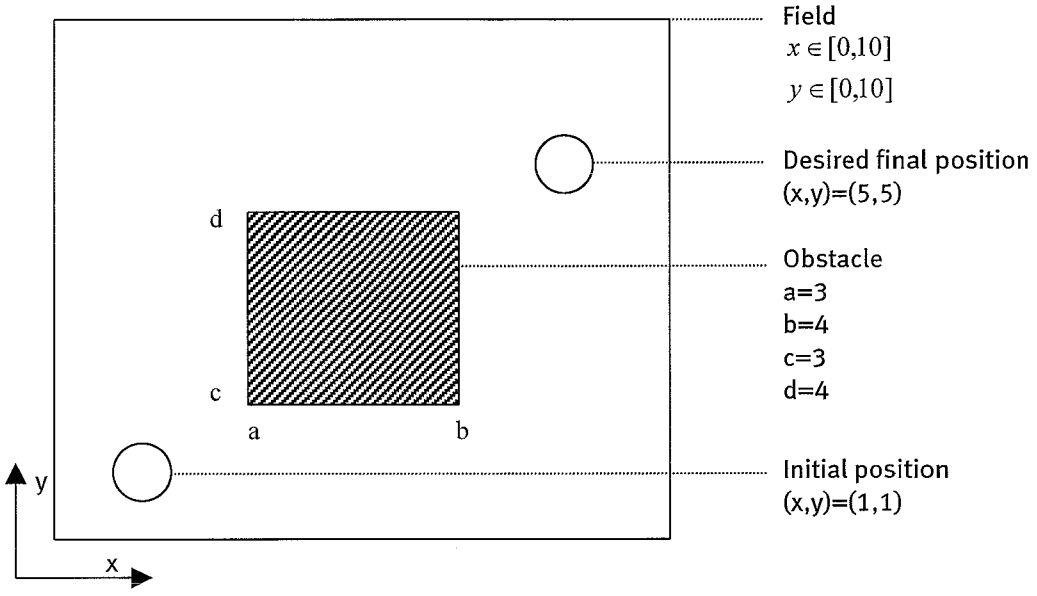


Figure 6.4: *Simulation configuration*

The robot is initially positioned at $(x, y) = (1, 1)$ and end state constraints are set to

$$(x_f, y_f) = (5, 5) \tag{6.42}$$

$$u_f = (0, 0) \tag{6.43}$$

$$\delta_f = \begin{pmatrix} \delta_{a1} \\ \delta_{a2} \\ \delta_{b1} \\ \delta_{b2} \\ \delta_{c1} \\ \delta_{c2} \\ \delta_{d1} \\ \delta_{d2} \\ \delta_{x1} \\ \delta_{x2} \\ \delta_{y1} \\ \delta_{y2} \\ \delta_{S1} \\ \delta_{S2} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \tag{6.44}$$

and z_f according to equation (6.2). (6.45)

A, B_1, B_2 and B_3 as mentioned in equation (6.14) are set to

```
A=zeros(2,2);
B1=zeros(2,2);
B2=zeros(2,14);
B3=[eye(2),eye(2)];
```

To describe the dynamics in area 1 (normal dynamics) and area 2 (obstacle) the following state and input matrices are chosen:

```
Adak1=eye(2);              % Discrete system matrix area 1
Bdak1=eye(2);              % Discrete input matrix

Adak2=eye(2);              % Discrete system matrix area 2
Bdak2=zeros(2,2);          % Discrete input matrix
```

The inputs are restricted to $u \in [-3, 3]$ and the prediction time is chosen to be $p = 4$ samples.

The weighting matrices as mentioned in equations (6.24) are defined in equation (6.47)

```
Q1=0.01;
QQ1=Q1*eye(nu*p);          % weight for u
Q2=0.00001;
QQ2=Q2*eye(ndelta*p);      % weight for delta(t)
Q3=1;
QQ3=Q3*eye(nz*p);          % weight for z(t)
Q4=1;
QQ4=Q4*eye(nx*p);          % weight for x(t)
```

6.5 SIMULATION RESULTS

With the described test configuration and a control time of $T = 5$ samples simulations has been conducted. The results of this experiment are presented in figure 6.5.

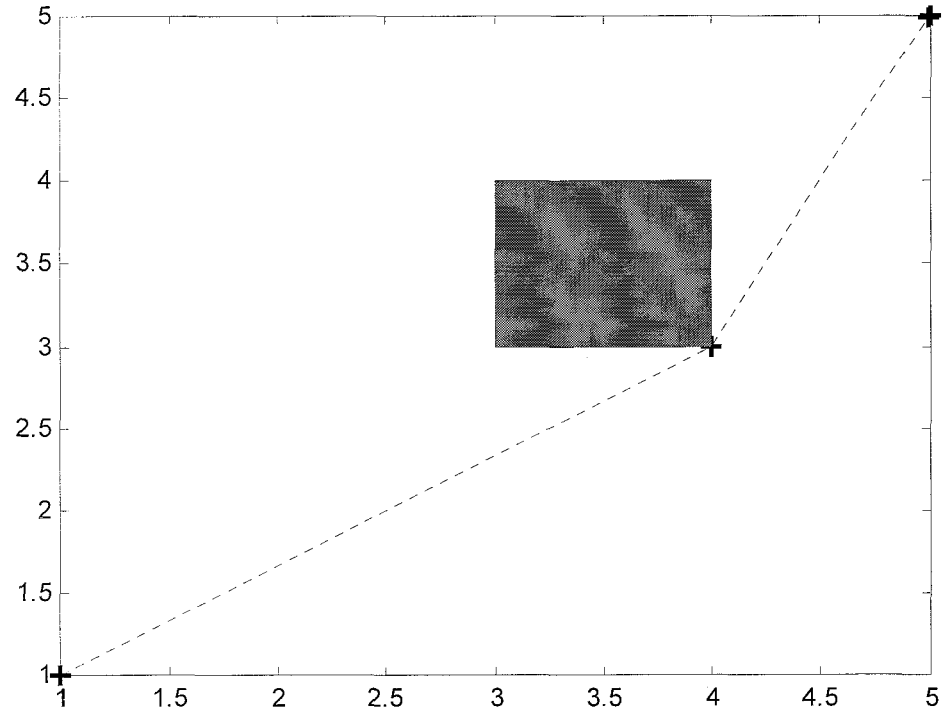


Figure 6.5: *Results of the example configuration, optimal path from (1,1) to (5,5) taking input and state constraints into account and avoiding one square obstacle.*

It is clear to see that the robot avoids the obstacle correctly and reaches the desired point. The designed MPC controller uses a hybrid (binary/continuous) optimisation. This routine is carried out by the MIQP software (version 1.02) designed by Bemporad et al. and uses the 'quadprog' algorithm from the optimisation toolbox of Matlab. The optimisation has proved to be very languid and quite critical in its configuration. Even a simulation of 5 steps with a relatively short prediction horizon of 4 samples, the optimisation takes approximately 4 hours on an Intel Pentium III processor (800 MHz).

In the considered Matlab script fourteen binary variables were used to describe the labelling of two areas and all binary variables were allowed to vary between 0 and 1. Some variables could be set to a fixed value. Because the robot will always be inside area 1 and therefore under d_1, above c_1, right of a_1 and left of b_1. The values of the corresponding δ can therefore be fixed to 1 (true). This could save significant amounts of computation time, but increases the 'risk' of introduced infeasibilities. Besides a different formulation of the problem another solver could be used to decrease the computation time, but this has not been tested during this project.

CONCLUSIONS AND RECOMMENDATIONS

It is very hard to construct a general valid framework for hybrid controllers using a Model Predictive Controller (MPC) with a hybrid optimisation for applications in emergency mode. This class of applications can be found in e.g. chemical plants and obstacle avoiding mobile robots. This last application has been chosen as an example system.
A lot of different approaches to obstacle avoidance are found in literature, some very fast, others very robust. A selection of methods is presented in chapter 2. It is difficult to compare two different methods, there most algorithms are especially designed for some specific situation or even one specific robot. Not only is it difficult to compare, it is very hard to design an obstacle avoidance algorithm in general terms, because the performance and algorithm is very dependent on the configuration, sensor inputs, constraints and environment.
An obstacle avoidance algorithm is designed in this report, which is based on hybrid system theory, using switching dynamics to describe obstacles, slippery areas and other objects. The model predictive controller optimises the trajectory of the robot by minimising a cost function taking several (end state) constraints into account.
Some remarks have to be made, the first one is about the calculation time. The designed algorithm is computationally very demanding. Even for the, in the emergency mode demanded, very short prediction times the computation times are vast. This problem can probably be (partially) solved by a more efficient labelling of the areas or a reduction of binary variables. Not all variables do necessarily have to vary during the simulations, some might be impossible (e.g. outside the working area) or some might be prohibited. These binary variables could be fixed within the simulation in a way that the optimisation does not have to try to vary these binary variables in a 'trail-and-error'-like optimisation. This method can contribute to a faster algorithm but increases the risk of introduced infeasibilities.
Another possible way to decrease the calculation times is to implement a faster optimisation algorithm. In this project only the mixed integer quadratic programming (MIQP) Matlab routine by [Bemporad,miqp.m] was used, but several other possibilities, like NAG or Fortran routines exist. Some caution on this point is necessary though, because these routines might have a slightly different definition of the problem, which is to be solved. Some of the existing optimisation routines solve a single-sided inequality equation; the used Matlab routine uses a double-sided inequality.
Another important possibility to increase the computational speed is to implement a so-called control horizon. This is a method, which is already successfully implemented in 'ordinary' MPC controllers but which is not used in this project, partially because the prediction horizon is already extreme short.
The second remark is about the restrictions that have to be imposed on the model. Because of the stringent format of the problems, which can be solved by the optimisation routine, the system has to be modelled in linear terms. This is quite stringent but on the other hand quite normal in MPC controlled systems. Filtering can probably offer some improvement in performance. In this report a very simple discrete linear system description is used to model a 'mobile robot'. Introduction of nonholonomic constraints and other extensions will impose new difficulties, which deserve special attention.
Because of the computational demands the algorithm is tested only with one square obstacle and a relatively short prediction horizon. It is clear to see in the results that the robot avoids the obstacle correctly and reaches its goal in an optimal sense. No special attention was given to the tuning of the designed MPC controller in this report, but it is

clear that most of the advantages are (partially) lost due to the very short prediction times. Model Predictive Control is a very powerful control strategy, mainly because of the possibility to base the current inputs on the predicted future responses of the system. By reducing the number of samples in the prediction horizon, this property will strongly be decreased in power.
There are several methods to model an obstacle. In most algorithms, the areas of obstacles are modelled as 'forbidden' areas. In this report the dynamics of the robot can be described as a function of the current area. By describing the dynamics of the robot in an area of an obstacle as an infinite difficult area, the optimal path of the robot will never go through this area and thus the robot will avoid this obstacle. In the current form there are some disadvantages of the algorithm; the algorithm only looks at the end points of each sample, in other words, if the robot can get 'over' an obstacle during one sample, the robot will go through the obstacle (partially) if this gains an optimal path. Only if an end point (after a sample) lie inside an obstacle area, the robot will not chose that specific path. Linearisation over the calculated trajectory can most likely solve this problem.
Model Predictive Control can most certainly be a very powerful solution for hybrid controllers in emergency mode, under the condition that the optimisation gets less computational demanding or can be executed on faster processors. Some attention has to be given to the construction of a correct cost function and end state constraints. If the designed MPC controller for hybrid systems is to be integrated in a hybrid controller, special attention has to be given to the switching problem.
The field of obstacle avoidance is a very interesting and active field within control theory. Looking at the current state of this field, other methods are a lot faster and even some very good real time implementations already exist. Considering the ever increasing processor speeds, Model Predictive Control (MPC) seems to be a promise for the (near) future, even for hybrid systems, but a lot of development work has to be carried out before this class of controllers can be applied in practise.

APPENDIX A – MATRIX DESCRIPTIONS FOR EXAMPLE

```
E1=[zeros(42,2);
Bdak1;
Bdak2;
-Bdak1;
-Bdak2;
eye(2);
-eye(2)];
```

```
E2=[(ma1-eps),zeros(1,13);
   Ma1,zeros(1,13);
   0,(ma2-eps),zeros(1,12);
   0,Ma2,zeros(1,12);
   0,0,(mb1-eps), zeros(1,11);
   0,0,Mb1,zeros(1,11);
   0,0,0,(mb2-eps),zeros(1,10);
   0,0,0,Mb2,zeros(1,10);
   zeros(1,4),(mc1-eps),zeros(1,9);
   zeros(1,4),Mc1,zeros(1,9);
   zeros(1,5),(mc2-eps),zeros(1,8);
   zeros(1,5),Mc2,zeros(1,8);
   zeros(1,6),(md1-eps),zeros(1,7);
   zeros(1,6),Md1,zeros(1,7);
   zeros(1,7),(md2-eps),zeros(1,6);
   zeros(1,7),Md2,zeros(1,6);
   1,0,1,zeros(1,5),-1,zeros(1,5);
   -1,zeros(1,7),1,zeros(1,5);
   0,0,-1,zeros(1,5),1,zeros(1,5);
   0,1,0,1,zeros(1,5),-1,zeros(1,4);
   0,-1,zeros(1,7),1,zeros(1,4);
   zeros(1,3),-1,zeros(1,5),1,zeros(1,4);
   zeros(1,4),1,0,1,zeros(1,3),-1,zeros(1,3);
   zeros(1,4),-1,zeros(1,5),1,zeros(1,3);
   zeros(1,6),-1,zeros(1,3),1,zeros(1,3);
   zeros(1,5),1,0,1,zeros(1,3),-1,zeros(1,2);
   zeros(1,5),-1,zeros(1,5),1,zeros(1,2);
   zeros(1,7),-1,zeros(1,3),1,zeros(1,2);
   zeros(1,8),1,0,1,0,-1,0;
   zeros(1,8),-1,0,0,0,1,0;
   zeros(1,10),-1,0,1,0;
   zeros(1,9),1,0,1,0,-1;
   zeros(1,9),-1,0,0,0,1;
   zeros(1,11),-1,0,1;
   zeros(1,12),-(Mx1-eps),0;
   zeros(1,12),-(My1-eps),0;
   zeros(1,12),0,-(Mx2-eps);
   zeros(1,12),0,-(My2-eps);
   zeros(1,12),mx1,0;
   zeros(1,12),my1,0;
   zeros(1,12),0,mx2;
   zeros(1,12),0,my2;
   zeros(1,12),-mx1,0;
   zeros(1,12),-my1,0;
   zeros(1,12),0,-mx2;
   zeros(1,12),0,-my2;
   zeros(1,12),(Mx1-eps),0;
   zeros(1,12),(My1-eps),0;
   zeros(1,12),0,(Mx2-eps);
   zeros(1,12),0,(My2-eps);
   zeros(4,14)];
```

```
E3=[zeros(34,4);
    eye(4);
    -eye(4);
    eye(4);
    -eye(4);
    zeros(4,4)];
```

```
E4=[-1,0;
    1,0;
    -1,0;
    1,0;
    1,0;
    -1,0;
    1,0;
    -1,0;
    0,-1;
    0,1;
    0,-1;
    0,1;
    0,1;
    0,-1;
    0,1;
    0,-1;
    zeros(26,2);
    Adak1;
    Adak2;
    -Adak1;
    -Adak2;
    zeros(4,2)];
```

```
E5=[(a1-eps);
   (Ma1-a1);
   (a2-eps);
   (Ma2-a2);
   (-b1-eps);
   (Mb1+b1);
   (-b2-eps);
   (Mb2+b2);
   (c1-eps);
   (Mc1-c1);
   (c2-eps);
   (Mc2-c2);
   (-d1-eps);
   (Md1+d1);
   (-d2-eps);
   (Md2+d2);
   1;0;0;
   1;0;0;
   1;0;0;
   1;0;0;
   1;0;0;
   1;0;0;
      eps;
      eps;
      eps;
      eps;
      zeros(4,1);
      -mx1;
      -my1;
      -mx2;
      -my2;
      Mx1;
      My1;
      Mx2;
      My2;
      -uxmin;
      -uymin;
      uxmax;
      uymax];
```

REFERENCES

[1] **Bemporad**, A., Morari, M. *Control of systems integrating logic, dynamics, and constraints.* Automatica volume 35, pp 407- 427, 1998.

[2] **Bissé**, E., Bentounes, M., Boukas, El-K. *Optimal Path Generation for a Simulated Autonomous Mobile Robot*, Autonomous Robots, pp 1-18, 1995.

[3] **Cavalier**, T. M. , Pardalos, P. M. and Soyster, A. L.*: Modeling and integer programming techniques applied to propositional calculus.* Computers Opns Res., 17(6):561--570, 1990.

[4] **Desai**, J.P., Kumar ,V. *Optimal Motion Plans for Cooperating Nonholonomic Mobile Manipulators in an Environment with Obstacles.*

[5] **Desai**, J.P., Kumar, V. *Nonholonomic motion planning for multiple mobile manipulators*, University of Pennsylvania.

[6] **Van Essen**, H.A. *Lecture notes Advanced Control Theory: Model Predictive Control,* Eindhoven University of Technology (The Netherlands), 2000.

[7] **Fraichard**, Th. *Trajectory Planning in a Dynamic Workspace: a 'State-Time Space' Approach*, Advanced Robotics, 1999.

[8] **Hayes**, J. P. *Introduction to Digital Logic Design.* Addison-Wesley Publishing Company, Inc., 1993.

[9] **Jacobsen**, D.H., Lele, M.M. *A Transformation technique for optimal control problems with a state variable inequality constraint.* IEEE Transactions on Automatic Control, vol. 14, pp 457-464, 1969.

[10] **Laubach**, S. Theory and Experiments in Autonomous Sensor-Based Motion Planning with Applications for Flight Planetary Microrovers, California Institute of Technology, 1999.

[11] **Liberzon**, D. and Morse, A.S. *Basic Problems in Stability and Design of Switched Systems*, IEEE Control Systems Magazine 19, 59-70, 1999.

[12] **Maciejowski** J.M. *Predicitve Control with contraints.* Harlow: Prentice Hall, 2002

[13] **Morari**, M. *Model Predicitve Control.* Prentice Hall, 1993

[14] **Murray**, R.M., Sastry, S.S. *Nonholonomic Planning: Steering using Sinusoids.* IEEE Transactions on Automatic Control, vol. 38, no. 5, pp 700-716, 1993. [20]

[15] **Rimon**, E. Kodischek, D. E. *Exact Robot Navigation Using Artificial Potential Functions.* IEEE Transactions on Robotics and Automation, 8(5 (October)):501-518, 1992.

[16] **Shiller**, Z., *On-line Sub-optimal Obstacle Avoidance.* Submitted to International Journal of Robotics Research, 1998.

[17] **Shiller**, Z., Gwo, Y.-R. *Dynamic Motion Planning of Autonomous Vehicles.* IEEE Transactions on Robotics and Automation, vol. 7, no. 2, pp 241 – 249. 1991.

[18] **Shiller**, Z. *Motion Planning for Mars Rover. (http://www.seas.ucla.edu/~shiller),* 1998.

[19] **Sundar**, S., Shiller, Z. *Optimal Obstacle Avoidance based on the Hamilton-Jacobi-Bellman Equation,* IEEE Transactions on Robotics and Automation, vol. 13 no. 2, pp 305 – 310, 1997.

[20] **Ulrich**, I., Borenstein, J. *VFH+:Reliable Obstacle Avoidance for Fast Mobile Robotics*, Proceedings of the 1998 IEEE International Conference on Robotics and Automation, Leuven (Belgium), pp 1572 – 1577, 1998.

[21] **Ulrich**, I., Borenstein, J. *VFH*:Local Obstacle Avoidance with Look-Ahead Verification*, 2000 IEEE International Conference on Robotics and Automation, San Francisco (CA), pp 2505 – 2511, 2000

[22] **Williams**, H. P. *Logical problems and integer programming*, 1977.

[23] **Zefran**, M. *Continuous Methods for Motion Planning. PhD thesis,* IRCS Report 96—34, University of Pennsylvania, 1996.

[24] **Zefran**, M., Desai, J.P., Kumar, V. *Continuous Motion Plans for Robotic Systems with Changing Dynamic Behavior*, Presented at 2nd Int.workshop on Algorithmic Foundations of Robotics, Toulouse (France), 1996.

PERSONAL MEMO