FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Invariance and Control of Autonomous Vehicles

Joana Isabel Lacerda da Fonseca Pinto Cardoso

MIEEC - Mestrado Integrado em Engenharia Electrotécnica

Supervisor: Karl Henrik Johansson (Prof.)
Royal Institute of Technology

Supervisor: João Tasso Sousa (Eng.)
Faculdade de Engenharia da Universidade do Porto

2010

Resumo

Actualmente, ambas as aplicações civis e militares de veículos aéreos não tripulados exigem um nível crescente de segurança. Um requisito comum em aplicações seguras é a capacidade de evitar obstáculos ou zonas inseguras. Esta especificação pode ser formulada como o problema de encontrar um controlador que mantenha o veículo dentro de um dado conjunto seguro, isto é, um conjunto em que se garante que o veículo não intersectará obstáculos desde que a sua trajectória não saia do mesmo. O problema de sintetizar controladores e conjuntos seguros não é de todo trivial, principalmente para dinâmicas e modelos do ambiente genéricos.

Este problema é estudado no contexto de uma missão cujo objectivo é alcançar uma posição final partindo de um dado estado inicial, evitando ao mesmo tempo quaisquer obstáculos ou zonas proibidas. Na nossa abordagem são usados modelos dinâmicos simplificados, assim como aproximações geométricas do ambiente. Começamos por desenvolver e provar o correcto funcionamento de um controlador de alto nível, baseado num autómato híbrido, cujo propósito é evitar obstáculos presentes no ambiente a analisar. Seguidamente, definimos e implementamos as condições de transição usando polígonos como obstáculos e, mais tarde, conjuntos baseados em *splines*, numa abordagem de complexidade geométrica crescente.

O problema é ainda estudado em dois cenários diferentes, o primeiro assumindo um conhecimento total do ambiente circundante, e o segundo, mais realista, onde o conhecimento do ambiente é adquirido de forma dinâmica à medida que o veículo viaja. O controlador desenvolvido não é máximo, no sentido em que o conjunto seguro que gera não é a união de todos os conjuntos seguros. No entanto, é facilmente implementável - um compromisso do óptimo pelo simples.

São utilizados exemplos numéricos para ilustrar a nossa abordagem. A solução, cumpridos determinados requisitos, encontra trajectórias viáveis para o sistema, capazes de manter o veículo fora das zonas inseguras.

Abstract

Currently, safety is becoming an increasingly important problem to both civilian and military unmanned aerial vehicle applications. One of the common requirements for any application is to avoid obstacles or other unsafe areas. We can formulate this requirement as a problem of finding a controller that keeps the vehicle inside a safe set, that is, a set for which we can guarantee that the vehicle does not enter any obstacle, as long as its trajectory remains in that set. The problem of synthesizing safe controllers and safe sets is not a trivial one, especially for generic dynamics and models of the environment.

We study this problem in the context of a mission where the main goal is to reach a final position given an initial location, while avoiding obstacles. In our approach we use a simplified dynamic model and geometric approximations of the environment. We start by devising and proving the correctness of a high-level collision avoidance controller based on a hybrid automaton. Then, we define and implement the transition conditions for both polygonal and spline-based obstacles, in increasing geometric complexity.

This problem is studied in two different scenarios, first assuming that the vehicle has a complete knowledge of the surrounding environment and then, in a more realistic fashion, letting the vehicle acquire its knowledge of the environment as it travels. The safe controller we develop is not maximal, in the sense that the set it generates is not the union of all safe sets. However the controller is easily implementable - a compromise between optimality and simplicity.

We illustrate our approach with numerical examples. Our solution, under the given assumptions, is able to find viable trajectories for the system, that is, trajectories that keep the vehicle from entering unsafe sets.

To my grandmother Mercedes, my parents Domingos and Margarida and my sister Ana
To Pedro

Contents

List of Figures

Abbreviations and Symbols

UAV	Unmanned Aerial Vehicle
AUV	Autonomous Underwater Vehicle
UGV	Unmanned Ground Vehicle
FRS	Forward Reachable Set
BRS	Backward Reachable Set

Chapter 1

Introduction

Currently, safety is becoming an increasingly important problem to both civilian and military unmanned aerial vehicle (UAV) applications. One of the common requirements for any application is to avoid obstacles or other unsafe areas. The collision avoidance problem takes place in numerous situations and usually does not appear isolated but as a part of a mission objective, such as to visit several waypoints or covering an area, together with an optimization problem, where the vehicle or formation has to minimize the time or distance or even the fuel it takes to complete the mission objective. Additionally, we face the collision problem, either in the case of avoiding a single obstacle or a set of them, or in the case of avoiding other vehicles or even both simultaneously. On the other hand, generic dynamics and models of the environment greatly increase the complexity of the problem, so some simplifications become necessary.

The avoidance problem can be formulated as a problem of finding a controller that keeps the vehicle inside a safe set, that is, a set for which we can guarantee that the vehicle does not enter any obstacle, as long as its trajectory remains in that set. We study this problem in the context of a type of mission where the main goal is to reach a final position from an initial location, while avoiding obstacles. At the same time, two different scenarios (in terms of the system's knowledge of the surrounding environment) are contemplated for this problem : one in which all the obstacles are known from the start, and another, more realistic, where the system's knowledge of the obstacles evolves with time as the system moves. We consider a simplified vehicle model, where we assume the vehicle travels at a constant altitude, which let us consider the world as two-dimensional.

Chapter 2

Related work

The collision avoidance problem takes place in numerous situations and usually does not appear isolated but as a part of a mission objective. Normally the main objective is to visit several waypoints, covering an area or minimizing some cost function as the distance or time. In addiction to these missions, we face the collision problem, either in the case of avoiding a single obstacle or a set of them, or in the case of avoiding other vehicles or even both simultaneously.

This scenario is described in [1], where the collision problem between UAVs appears as one of the two constraints of a control objective. The objective is to maximize the regions visited by a team of UAVs, while minimizing the visits to dangerous zones. The other constraint is that the communication network remains connected all the time. To avoid collisions, each vehicle has a rectangular exclusion region around it where no other vehicle can enter. This rectangular region is set such that the space between vehicles is greater than a safety distance in the four directions X, -X, Y and -Y.

In [2] a minimum time formulation is made, again combined with the collision avoidance constraint. Here, each aircraft is supposed to visit a number of waypoints and the order of those visits is determined by the optimization problem of minimizing the overall flight time.

In [3] the scenario is similar to [1] and [2] with the exception of the optimization problem which, in this case, is to find the optimal path between two states, for a single vehicle or multiple vehicles, that minimizes the fuel and/or time while avoiding collisions with each other or with fixed or moving obstacles.

A different approach is shown in [4], where a decentralized control law, based on the Navigation function method, is presented. The idea of this method is to create a global potential field, where an attracting potential is placed at the target point while repulsive potentials, pushing the vehicle away, are placed at both the starting position and the obstacles. The vehicles nagivate through these gradient fields towards the target point. Gyroscopic forces are used to accomplish the control objective while avoiding collisions with possible obstacles or other agents. The use of gyroscopic forces for steering does not affect the global potential function or even change the energy of the

system, making their use appropriate for the avoidance problem. A braking force is also used to decelerate the vehicles when these are moving too fast and getting too close to an obstacle or another vehicle.

Another common technique is the use of reachable sets to solve the collision avoidance problem. Having the backward reachable set (BRS) of an obstacle makes it simple to determine if the trajectory of the vehicle is safe by checking if it does not enter that set. In [5] computational methods are developed in order to determine in what states a system is allowed to be, and this information is used to avoid unsafe sets. The BRS of a continuous dynamic system is computed using level set methods to solve the time dependent Hamilton-Jacobi-Isaacs partial differential equation. Similar work is done in [6] where the authors develop a method, based on the level set techniques, which determines a representation of the boundary of the reachable set departing from an initial set.

In a different approach, Cataldo in [7] also uses level set methods to determine boundary zones around restricted airspace or no-fly zones stored in a 3D database on-board each aircraft - in what is called "soft walls" strategy. Instead of completely removing the pilot input when the aircraft is moving towards a no-fly zone, the controller adds a varying bias to the pilot's input, to ensure that the aircraft does not enter in that zone.

Based on [5, 6] and [7], Frew and Sengupta developed an obstacle avoidance approach in [8], where the reachable set computation indicates the last moment at which the aircraft can start an avoidance maneuver. The avoidance problem appears at a formation planner level of a multi-objective, hierarchical architecture for a collaborative control of a formation of UAVs.

There exists a large amount of literature addressing the collision avoidance problem, not only with UAVs but with ground and underwater vehicles too. However, because of the stall conditions, a minimum velocity constraint is imposed to the UAVs, and so most of the results for Autonomous Underwater Vehicles (AUVs) and Unmanned Ground Vehicles (UGV) can not be applied to UAVs. Since the full dynamics of an UAV is complex, a simplified model is usually considered to solve these types of problems, assuming the vehicle travels at a constant altitude. These assumptions also appears in some of the previous references. As it is mentioned in [9], UAVs often operate at a fixed altitude because of limited sensor range and the threat of enemy radar. So, assuming that the vehicle moves in two dimensional scenario, the mathematical complexity is reduced while maintaining a realistic case.

Chapter 3

Mathematical background

There are several concepts that are fundamental for the understanding of our work. For this reason, we will give a briefly background that follows [14] and [15] closely.
One system can be interpreted as a function that relates a set of input signals, a set of output signals and the function itself.

$$\mathscr{F} : Input\ Signals \rightarrow Output\ Signals$$

These signals can be continuous in time - continuous time systems - or they can be discrete time signals - discrete time systems.

3.1 Continuous time systems

Definition 1 (Controlled continuous time systems) *A controlled continuous time system is described by*

$$\begin{cases} \dot{x}(t) = f(t, x(t), u(t)) \\ y(t) = h(x(t)) \end{cases} \tag{3.1}$$

where $f : \mathbb{R} \times \mathbb{R}^n \times \mathbb{R}^u \rightarrow \mathbb{R}^n$, $h : \mathbb{R}^n \rightarrow \mathbb{R}^q$, $x(t) \in \mathbb{R}^n$ *is the system state,* $u(t) \in U(t) \subset \mathbb{R}^u$ *is our control input,* $U(t)$ *is the set of all admissible controls and* $y(t) \in \mathbb{R}^q$ *is the output.*

In continuous time, we assume that f and h are Lipschitz functions and that $u(t)$ is continuous. This means that for all $x(0) \in \mathbb{R}^n$ and $u(t) \in U(t)$, the system admits an unique solution $x(t)$ defined on $\mathbb{R}^+$.
Since $f(t, x(t), u(t))$ is a Lipschitz function in x, we can represent the system as:

$$\dot{x}(t) \in F\left(t, x(t)\right) \subset \mathbb{R}^p \tag{3.2}$$

where $F\left(t, x(t)\right) := \left\{s : s = f\left(t, x(t), u(t)\right), u(t) \in U(t)\right\}$. This is a differential inclusion, where $F(t, x(t))$ maps $(t, x(t))$ onto the set of admissible velocities at $(t, x(t))$. It is important to mention that for local controllability, the condition $0 \in int\left\{F\right\}$ is necessary.
Typically, f is a differential equation that relates the system state and its evolution with the control signal $u(t)$.

Definition 2 (Differential equation) *Differential equations are equations that relate a function $x(t)$, its derivatives and the independent variable t [16].*

$$\mathscr{R}(t, x, x', x'', ..., x^k) = l(t) \tag{3.3}$$

with a function $l(t)$ which only depends on t. The order of a differential equation is given by the highest derivative of $x(t)$ that appears in $\mathscr{R}$.

The equations where x depends on an independent variable t and that can be written as in 3.3, are called ordinary differential equations.

Definition 3 (Solution of a ordinary differential equation) *The solution of a ordinary differential equation of the k^{th} order, defined in an interval $I = [a, b]$ (where a and b can be $-\infty$ and $+\infty$, respectively), is a continuous function, with the derivatives up to the k^{th} order defined in that interval, and satisfying definition 2.*

3.2 Discrete time systems

Definition 4 (Discrete time systems) *A controlled discrete time system is described by*

$$\begin{cases} x(n+1) = f(n, x(n), u(n)), \forall n \in \mathbb{Z} \\ y(n) = h(x(n)) \end{cases} \tag{3.4}$$

where $f : \mathbb{Z} \times \mathbb{R}^n \times \mathbb{R}^u \to \mathbb{R}^n$, $h : \mathbb{R}^n \to \mathbb{R}^q$, $x(n) \in \mathbb{R}^n$ is the system state, $u(n) \in \mathscr{U}(n) \subset \mathbb{R}^u$ is our control input, $U(n)$ is the set of all admissible controls and $y(n) \in \mathbb{R}^q$ is the output.

Normally, these systems are described by difference equations.

Definition 5 (Difference equation) *Consider a sequence $\{a_n\}$ of real numbers. The first difference of the sequence, $d(a_n)$, is the difference $a_n - a_{n-1}$. The second difference $d^2(a_n)$ is*

$d(a_n) - d(a_{n-1})$, which is equal to $a_n - 2a_{n-1} + a_{n-2}$. Generally, the k^{th} difference $d^k(a_n)$ is given by $d^{k-1}(a_n) - d^{k-1}(a_{n-1})$. A difference equation is an equation involving a_n and its differences ([17]).

3.3 State Machines

As we have mentioned previously, some systems can be described by differential equations - continuous time systems or difference equations - discrete time systems. Both these systems are time-driven, and can be represented using state-space models.
As opposed to these, we can also have event-driven systems which can, for example, be represented by state machines. These are defined as follows.

$$State\ Machine = \begin{cases} States \text{ - the finite set of state space} \\ Inputs \text{ - all infinite sequences of input values} \\ Outputs \text{ - all infinite sequence of output values} \\ Update \text{ - the transition function that updates the state} \\ Initial\ State \text{ - the first state to be considered} \end{cases}$$

According to the state machine operation, given a specific input signal, the correspondent output signal, using for that the update function and the initial state, is:

$$s(0) = Initial\ State; \tag{3.5}$$

$$\forall n \geq 0 : y(n) = update(s(n), x(n)), s(n+1); \tag{3.6}$$

$$\tag{3.7}$$

3.4 Hybrid systems

There are some systems that, due to their dynamics, cannot be characterized using the previous concepts separately. This systems are called hybrid systems.
A hybrid system is formally modeled by an hybrid automaton which defines the evolution of the hybrid system state. The hybrid automaton is characterized as follows ([18]).

$$Hybrid\ Automaton = \begin{cases} \mathbb{Q} \equiv \text{set of discrete states;} \\ X \equiv \text{continuous state space;} \\ Init \subseteq \mathbb{Q} \times X \equiv \text{initial states;} \\ f : \mathbb{Q} \times X \to X \equiv \text{vector field;} \\ \varphi : \mathbb{Q} \times X \to \mathbb{Q} \equiv \text{discrete transition;} \\ \rho : \mathbb{Q} \times X \to X \equiv \text{reset map.} \end{cases}$$

The solution of a hybrid automaton, $\chi = (\tau,q,x)$, is a pair of signals:

- Time trajectory τ: it is the time on which the solution is defined
- State trajectory (q,x): state evolution defined on τ of the hybrid automaton

3.5 Set theory concepts

This section introduces some relevant concepts from set theory.

Definition 6 (Open set) *A set $\mathscr{S}$ is an open set if every point in $\mathscr{S}$ has a neighborhood lying in the set.*

Definition 7 (Closed set) *A closed set is a set whose complement is open.*

Definition 8 (Convex set) *A set is said to be convex if, given any two points in the set, all the points along the line that connects them are also in the set.*

Throughout this work, we denote the interior of a set $\mathscr{S}$ as $int\{\mathscr{S}\}$ and its boundary as $\partial\mathscr{S}$. Another important definition that will be a key aspect in our approach is the distance to a given set.

Definition 9 (Distance to a set) *Consider a generic point P and a set $\mathscr{S}$. The distance between P and $\mathscr{S}$ is given by:*

$$d_{\mathscr{S}}(P) = \min_{s\in\mathscr{S}} \|P - s\| \tag{3.8}$$

where $\|\cdot\|$ denotes the euclidean norm.

When specifying the desired behavior of a particular system, with respect to a closed set $\mathscr{S}$, which is the one we have in hands, the following properties are used.

- Safety - the trajectories of the system do not enter $\mathscr{S}$.
- Invariance - the trajectories of the system do not leave $\mathscr{S}$.
- Attainability - the trajectories of the system enter $\mathscr{S}$.
- Performance - evaluates the trajectories of the system using cost functions for invariance, attainability and safety problems.

3.6 Reach sets

A reachable set can be considered both forward and backwards in time, denoted by FRS and BRS respectively. $FRS(\tau, x_0, t_0)$ is defined as the set of all possible states that the system can enter within a given time τ, departing from a specific position x_0, at time t_0.

Definition 10 (Forward reachable set starting at a given point) *Suppose the initial position and time (x_0, t_0) are given. The $FRS(\tau, x_0, t_0)$ of system 3.1 at time $\tau \geq t_0$ starting at position and time (x_0, t_0) is given by:*

$$\mathscr{R}[\tau, t_0, x_0] = \bigcup \{x(\tau), u(s) \in U(s), s \in (t_0, \tau]\} \tag{3.9}$$

If instead of a starting point we have a starting set, the definition of forward reachable set is similar to the previous one.

Definition 11 (Forward reachable set starting at a given set) *The forward reach set at time $\tau > t_0$ starting from a set χ_0 is defined as :*

$$\mathscr{R}[\tau, t_0, \chi_0] = \bigcup \{\mathscr{R}[\tau, t_0, x_0] | x_0 \in \chi_0\} \tag{3.10}$$

The $BRS(\tau, \chi_f)$ is defined as the set of all points which, using an admissible control, it is possible to reach the target χ_f in τ seconds.

Definition 12 (Backward reachable set) *The backward reach set of a set χ_f at time $\tau > t_0$ is given by:*

$$\mathscr{R}[\tau, \chi_f] = \bigcup \{x(\tau) : \exists u(s) \in U(s) : x(\tau) \in \chi_f, s \in (\tau, t_f]\} \tag{3.11}$$

3.7 Invariance

In this section we will present some definitions related to the concept of invariance .

Definition 13 (Weak invariance) *The set $\mathscr{S}$ is weakly invariant if for a controlled differential equation $\dot{x}(t) \in F(t, x(t))$ there exist controls such that a trajectory starting inside $\mathscr{S}$ remains inside $\mathscr{S}$.*

Sometimes it is necessary to guarantee that the system never leaves the set $\mathscr{S}$. The intuition is that all feasible controls, for any point of $\partial\mathscr{S}$, point to the interior of the set $\mathscr{S}$. From this notion comes the definition of strong invariance.

Definition 14 (Strong invariance) *The set $\mathscr{S}$ is strongly invariant if for a differential inclusion $\dot{x}(t) \in F(t,x(t))$ any trajectory starting inside $\mathscr{S}$ never leaves $\mathscr{S}$, for all admissible controls.*

3.8 Avoidance, safe and maximal safe set

We define the avoidance or 'bad' set $\mathscr{Q}$ as the set of states that we do not want the system to enter.

Definition 15 (Safe set) *Given a system as described by equation 3.1, a set $\mathscr{S}_0$ is said to be safe with respect to an avoidance set $\mathscr{Q}$, if and only if, for all $\tau \geq t_0$:*

$$\mathscr{R}[\tau, t_0, \mathscr{S}_0] \cap \mathscr{Q} = \emptyset \tag{3.12}$$

where $\mathscr{R}(\cdot)$ denotes the system's reach set (recall definition 11). The largest set $\mathscr{S}_0$ for which this condition holds is called the maximal safe set, $\mathscr{S}^*$.

3.9 Curves

Now that we already introduced the main definitions concerning our work, let us review some concepts related to curves, as we will use these to define our avoidance sets.
Given a curve $H \subset \mathbb{R}^n$, $n = 1,2,\ldots$, a parametric representation of H is a vector valued function, defined over a real interval ([21]):

$$H : I \subset \mathbb{R} \to \mathbb{R}^n \tag{3.13}$$

$$t \to H(t) = (H_1(t), \ldots, H_n(t)) \tag{3.14}$$

Related to the curves there are some important properties that we will briefly define.
Consider $H : I \to \mathbb{R}^n$ as a parametric representation of a differentiable curve $(H \in C^2)$ and suppose that $H'(t) \neq 0$ for all $t \in I$.

Definition 16 (Tangent vector) *The tangent vector to each point of the curve parametrized by H is given by :*

$$T(t) = \frac{H'(t)}{\|H'(t)\|} \tag{3.15}$$

Definition 17 (Normal vector) *The vector normal to the curve is given by :*

$$N(t) = \frac{T'(t)}{\|T'(t)\|} \tag{3.16}$$

for $T'(t) \neq \emptyset$.

The geometric interpretation of definitions 16 and 17 is shown in figure 3.1.

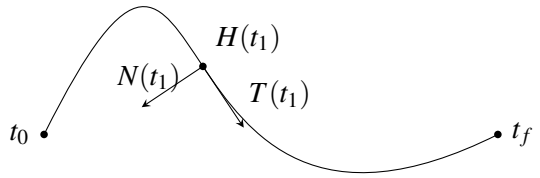


Figure 3.1: Tangent and normal vector at instant t_1

Another important feature of a curve is its arc length.

Definition 18 (Arc length) *Consider* $H : I \to \mathbb{R}^n$, $I = [a,b]$ *and* $C^1(\mathbb{R}^n, \mathbb{R})$. *The arc length of the curve parametrized by H between* $t \in [a,b]$ *is :*

$$L = \int_a^b \|H'(t)\| dt \tag{3.17}$$

Consider that our curve is parametrized with respect to the arc length s. The unit tangent vector, T, has a constant length for all points of the curve, changing only in direction. The relation between this change in direction and the arc length is given by the curvature.

Definition 19 (Curvature) *A curve curvature parametrized with respect to s is given by:*

$$k = \left\| \frac{dT}{ds} \right\| \tag{3.18}$$

Given a generic parameter t, the curvature's expression becomes:

$$\left\| \frac{dT}{ds} \right\| = \frac{1}{\|H'(t)\|} \left\| \frac{dT}{dt} \right\| \tag{3.19}$$

Definition 20 (Curvature radius) *The curvature radius, ρ is the inverse of the curvature.*

$$\rho = \frac{1}{k} \tag{3.20}$$

3.10 Splines

Definition 21 (Cubic spline) *A cubic spline function $S(s)$ is defined by:*

$$S = \{x \in \mathbb{R}^2 : x = KS_0(s), s \in [s_-, s_+]\} \tag{3.21}$$

where $K = \begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \end{bmatrix}$ *and* $S_0(s) = [s^3 \; s^2 \; s \; 1]^T$.

3.11 Vehicle model

We assume that the UAV travels at constant linear velocity and altitude and hence we consider an unicycle model of the UAV for synthesizing the avoidance controller.

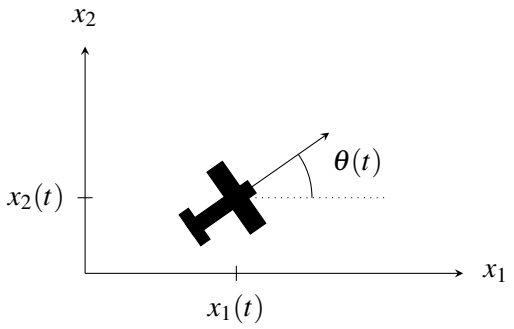


Figure 3.2: Unicycle model

This model has the following dynamics:

$$\begin{cases} \dot{x}_1(t) &= U \cdot \cos(\theta(t)) \\ \dot{x}_2(t) &= U \cdot \sin(\theta(t)) \\ \dot{\theta}(t) &= \omega(t) \\ \omega(t) &= \omega_{cmd} \cdot (t - T_d), \quad |\omega_{cmd}| \leq \omega_{max} \end{cases} \tag{3.22}$$

where $[x_1, x_2]^T$ is the position of the vehicle, θ is its orientation, U and ω are its linear and angular speeds, respectively, ω_{cmd} is the angular speed command, ω_{max} is the maximum angular speed and T_d is a time delay caused by the sensor readings, system processing and roll dynamics. It is important to note that the linear velocity should always be kept above the minimum steady controllable flight speed, so as to prevent the aircraft from stalling, and below the maximum structural cruising speed, in order to comply with the aircraft's structural limits.

Chapter 4

Problem description

4.1 Problem background

Our problem of obstacle avoidance appears in a usual mission context, where the vehicle has to move from an initial position to a final point, possibly while under additional mission objectives (such as distance or time minimization). We assume that there already exists a controller that is able to fulfill the mission objectives, considering that there are no obstacles.
Our goal is to develop an avoidance controller and, since the mission controller is provided, it is also necessary to develop a higher level controller that is able to decide which controller should be active. This means that the system will have a supervisory architecture, as it may be necessary to switch between the avoidance and mission controllers (see figure 4.1).

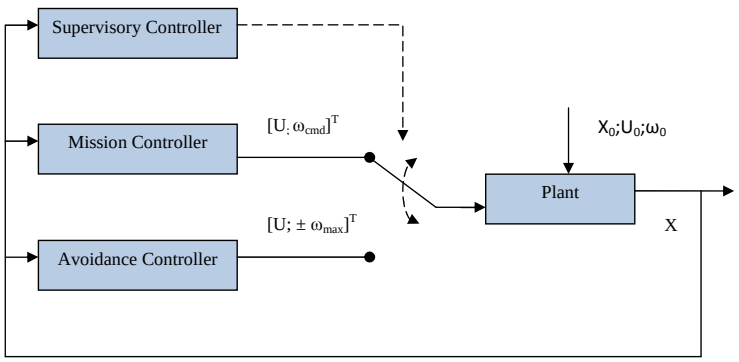


Figure 4.1: Controller architecture

At the same time, two different scenarios (in terms of the system's knowledge of the surrounding environment) should be contemplated for this problem : one in which all the obstacles are known

from the start, and another, more realistic, where the system's knowledge of the obstacles evolves with time as the system moves.

4.2 The mission controller

The controller that we present in this section represents a simple example of a mission controller, what we call the "*point-and-go*" controller. Notice, however, that it is presented here not as a part of the problem but as the mission controller that will be used throughout the rest of this work.
According to the system's state, as well as the target position, we start by defining two intervals, for which turning right or left will drive the system to achieve a required heading, and after which it will travel towards the target. Consider, for example, a positive orientation of the system, represented by $\hat{\theta}$ in figure 4.2, as well as the intervals $I_1 = [\theta, \pi] \cup [-\pi, \theta - \pi]$ and $I_2 = [\theta - \pi, \theta]$.

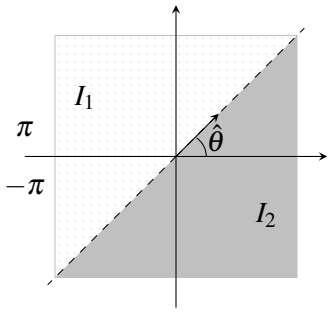


Figure 4.2: Interval illustration for a positive orientation of the system

If the final position is such that the desired heading is in I_1 then the system turns to the left. Otherwise, the system turns to the right.
If, however, the system has a negative orientation, as presented in figure 4.3, two different intervals are set, $I_3 = [\theta, \theta + \pi]$ and $I_4 = [\theta + \pi, \pi] \cup [-\pi, \theta]$, such that the system turns to the left if the desired heading is in I_3 and right otherwise.

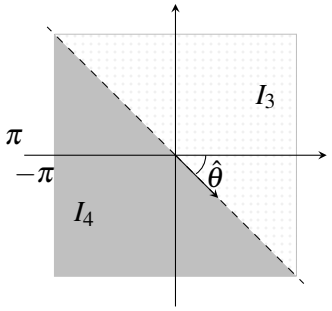


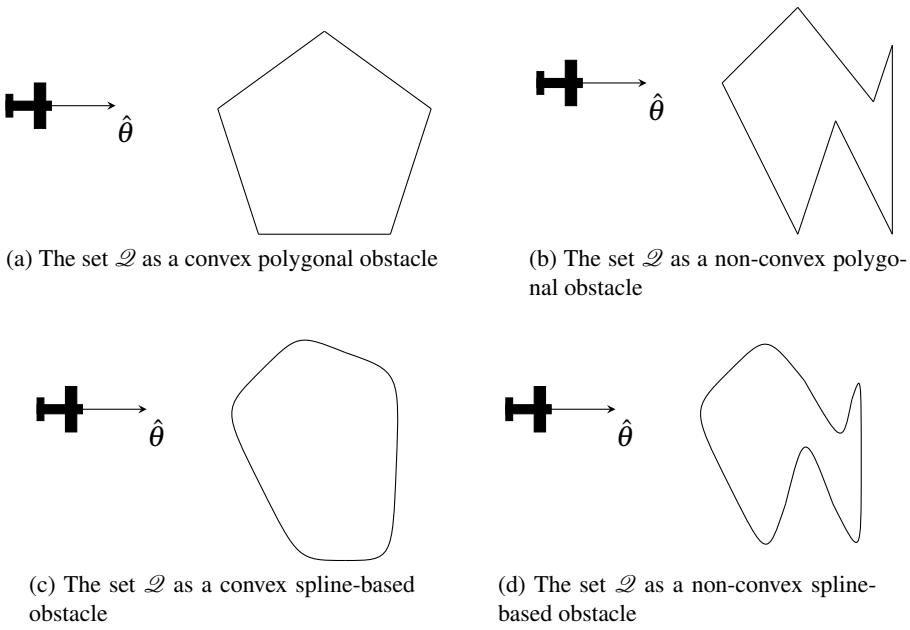
Figure 4.3: Interval illustration for a negative orientation of the system

4.3 Problem formulation

In this section we present a formal description of our problem.
Consider the unicycle model as described in 3.22 and an unsafe closed set $\mathscr{Q}$, describing a given obstacle, no-fly zone or unsafe region. We want to find a safe controller $u = \Gamma(\cdot)$ that guarantees that the vehicle never leaves a (certain) subset $\mathscr{S}$ of the system's maximal safe set $\mathscr{S}^*$ as it is going from an initial state to a final state:

$$\textit{Find } u = \Gamma(\cdot) \textit{ such that } x(t) \in \mathscr{S} \subset \mathscr{S}^*, \forall t \in [t_1, t_2] \tag{4.1}$$

We let the set $\mathscr{Q}$ be defined by polygons and spline-based sets, as the ones presented in figure 4.4.



(a) The set $\mathscr{Q}$ as a convex polygonal obstacle

(b) The set $\mathscr{Q}$ as a non-convex polygonal obstacle

(c) The set $\mathscr{Q}$ as a convex spline-based obstacle

(d) The set $\mathscr{Q}$ as a non-convex spline-based obstacle

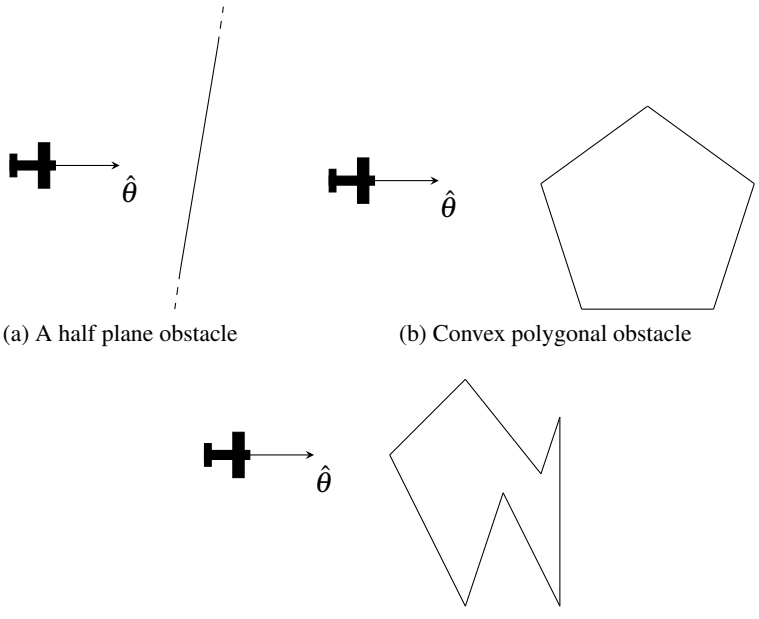
Figure 4.4: Different examples of the avoidance set $\mathscr{Q}$

Chapter 5

A completely known world

5.1 Polygonal obstacles

In this section we will study how to solve the avoidance problem given a polygonal obstacle.
We start by studying how to avoid a shifted half plane (or simple wall) as shown in figure 5.1(a). In here, we introduce important definitions as well as the controller that we will use throughout our approach. We will then extend these results to solve the problem with convex polygonal obstacles as the one presented in figure 5.1(b) and finally with non-convex polygonal obstacles as shown in figure 5.1(c).



(a) A half plane obstacle

(b) Convex polygonal obstacle

(c) Non-convex polygonal obstacle

Figure 5.1: Different type of polygonal obstacles

5.1.1 Simple Wall

Definition 22 (Shifted half plane or simple wall) *A shifted half plane is defined by two vertices V_0 and V_1 and a normal vector $\hat{R}$ pointing towards the interior of the half plane.*

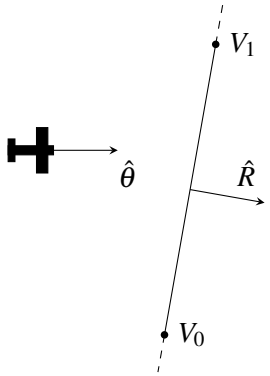


Figure 5.2: Shifted half plane

Our main goal is to avoid the collision between the vehicle and the obstacle. Thus it is important to define what we mean by "collision".

Definition 23 (Collision) *The vehicle has collided with the obstacle if the following condition holds:*

$$x \in \mathscr{Q} \tag{5.1}$$

where x is the vehicle's position and $\mathscr{Q}$ the obstacle.

In order to avoid collisions we introduce the two avoidance maneuvers that drive the vehicle away from the obstacle, turning the airplane in the desired direction.

Definition 24 (Avoidance maneuvers) *Given an arbitrary position and orientation of the vehicle, $x = [x_1, x_2, \hat{\theta}]^T$, we define two avoidance maneuvers as the two symmetric semi-circular maneuvers departing from x with radius equal to the minimum turning radius of the vehicle, r_c. These two maneuvers are denoted by x_{left} and x_{right}, and are obtained by setting $\omega = \omega_{max}$ and $\omega = -\omega_{max}$, respectively.*

The centers of these circular maneuvers are called X^{CL} and X^{CR} and are presented in figure 5.3.

Avoidance maneuvers should only be used at the right time, i.e. when there is no other choice for the vehicle but to escape from the obstacle. In other words, the vehicle should engage an avoidance maneuver every time that, by not doing so, will place it in a collision course with the obstacle.

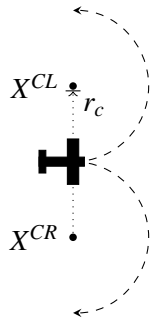


Figure 5.3: Avoidance maneuvers

Definition 25 (Viable trajectory) *A trajectory is viable if and only if it does not collide with the obstacle, i.e. if definition 23 does not hold for any state along the trajectory.*

So, the vehicle should travel freely while at least one avoidance maneuver is viable. Thus, when the vehicle is moving towards the obstacle, we test its state continuously, analyzing the viability of the avoidance maneuvers starting from the state at the next time step.

Definition 26 (Collision risk) *We say that there is a collision risk if the vehicle is going towards the obstacle and its distance to it is less than four times the minimum turning radius. Defining* $\hat{\theta} = [\cos(\theta), \sin(\theta)]^T$*, we can state this as the following condition.*

$$\hat{\theta}^T \hat{R} > 0 \ \wedge \ d_{\mathscr{Q}}(x) \leq 4r_c \tag{5.2}$$

Now that we have presented the main definitions for our problem, let us present our control strategy through our hybrid automaton, presented in figure 5.4.

The controller starts in a standby mode. If the vehicle is approaching the obstacle (condition C), i.e. if definition 26 holds, the controller transits to the test mode, testing the viability of the left and right avoidance maneuvers separately. Should *one* of the maneuvers fail (condition L or R), there is still a way to avoid the obstacle, and so the controller transits to the correspondent fail mode. If both maneuvers become inviable simultaneously, or if in fail mode, the remaining maneuver becomes inviable, the controller starts the correspondent avoidance maneuver. Our approach is summarized with theorem 1.

Theorem 1 (Safe avoidance controller) *Consider the unicycle model described by system 3.22 and controlled by the hybrid automaton shown in figure 5.4. Assuming that all the avoidance maneuvers departing from the vehicle's initial state* $x(t_0) = X_0$ *are viable, the following property holds:*

$$x \notin \mathscr{Q} \, , \, \forall t > t_0 \tag{5.3}$$

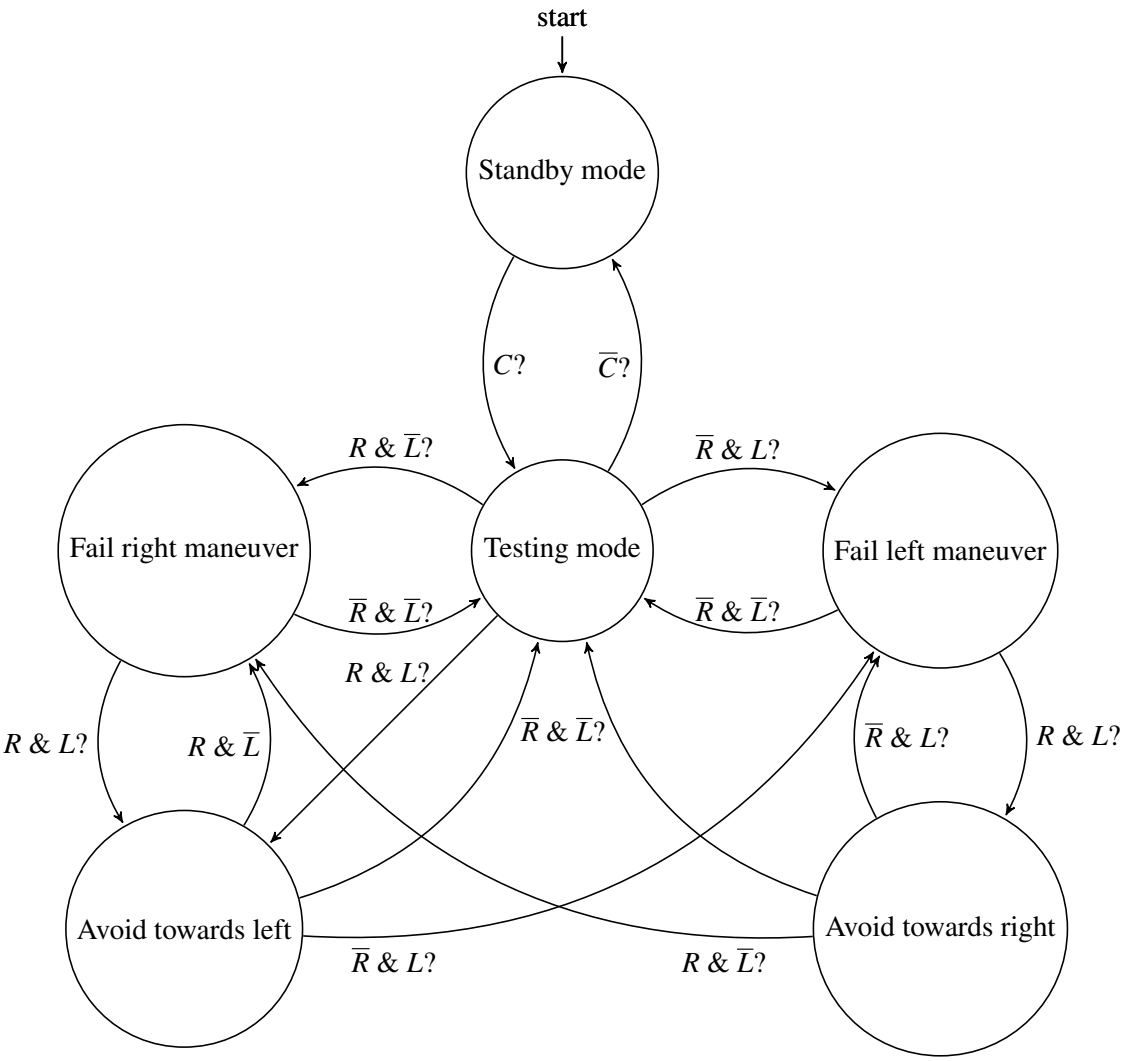


Figure 5.4: Hybrid automaton for the avoidance controller

Proof. Let $x(t_0) = [x^c(t_0), \hat{\theta}(t_0)]^T$ be the actual state of the vehicle where $x^c(t_0) = [x_1(t_0), x_2(t_0)]^T$. Assume that all the avoidance maneuvers departing from the vehicle's initial state are viable. Define:

$$d_{\mathscr{Q}}(X^c) = \min_{q \in \mathscr{Q}} \|X^c - q\| \tag{5.4}$$

as the distance between the vehicle and the obstacle $\mathscr{Q}$.
Recall that X^{CR} and X^{CL} are the centers of right and left avoidance maneuvers, respectively.
Let t_1 be the first instant for which the following condition holds, for a given $\varepsilon > 0$.

$$R : d_{\mathscr{Q}}(X^{CR}(t)) < r_c + \varepsilon \tag{5.5}$$

t_1 is the instant when the right avoidance maneuver becomes inviable and also the instant when the controller transits for the fail right maneuver state.
Assume that the following condition does not hold for any instant before and including t_1.

$$L : d_{\mathscr{Q}}(X^{CL}(t)) < r_c + \varepsilon \tag{5.6}$$

Let t_2 be the first instant when condition 5.6 holds. Assume that condition 5.5 also holds between t_1 and t_2.
So, in t_2 we guarantee that,

$$d_{\mathscr{Q}}(X^{CL}(t_2)) > r_c \tag{5.7}$$

for a given $\varepsilon > 0$. Therefore at t_2 the controller switches to the avoid towards left state, starting the left avoidance maneuver, avoiding the obstacle.
We can easily get the same results starting the other way around, i.e. when the left avoidance maneuver is the first to fail. ■

For this controller to work, we must implement the L and R conditions. One possible solution is to solve the point projection problem. This problem applied to the simple wall, consists in, given a straight line L defined by two points, l_1 and l_2, and a point P that does not belong to L, compute the point C in L that is closest to P as well as the distance between P and C (see figure 5.5).

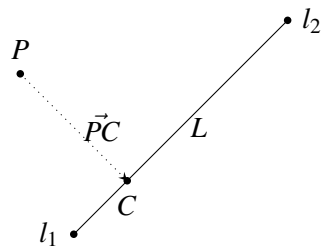


Figure 5.5: Point projection problem for a straight line

We should make an important remark about this problem: the vector $\vec{PC}$ is perpendicular to the line, and whatever the point P is, vector $\vec{PC}$ will have the same direction, since a line only has one normal direction. This direction leads us to the line's closest point.
The conditions L and R compare the norm of the vector $\vec{PC}$ with a fixed value r_c. Thus, instead of solving the point projection problem explicitly, we can consider a point X_T, that is r_c meters away from the point P along the direction of the vector $\vec{PC}$ and check if X_T is in the semi plane.

Definition 27 (Test point) *Consider an avoidance maneuver described by definition 24 and a simple wall as in definition 22. We define the maneuver's test point as the point along the maneuver which is closest to the simple wall.*

This way, we only need to analyze the test points, since if these are not inside the unsafe set, then the corresponding avoidance maneuver will not intersect that set. Considering the wall's normal vector $\hat{R}$ is known, we explain next how to find the point along the avoidance trajectory that is closest to a given line segment.
We define the vector $\vec{CL}$ as the vector that points from the vehicle's position x, to the center of the left avoidance maneuver, X^{CL}. Analogously, $\vec{CR}$ is the vector that points from x to the center of the right avoidance maneuver X^{CR}:

$$\vec{CL} = \begin{bmatrix} r_c \cos(\theta + \frac{\pi}{2}) \\ r_c \sin(\theta + \frac{\pi}{2}) \end{bmatrix} = \begin{bmatrix} -r_c \sin(\theta) \\ r_c \cos(\theta) \end{bmatrix} \tag{5.8}$$

$$\vec{CR} = \begin{bmatrix} r_c \cos(\theta - \frac{\pi}{2}) \\ r_c \sin(\theta - \frac{\pi}{2}) \end{bmatrix} = \begin{bmatrix} r_c \sin(\theta) \\ -r_c \cos(\theta) \end{bmatrix} \tag{5.9}$$

$$X^{CL} = x + \vec{CL} \tag{5.10}$$

$$X^{CR} = x + \vec{CR} \tag{5.11}$$

where X^{CL} and X^{CR} and the centers of the left and right avoidance maneuvers, respectively. We illustrate this in figure 5.6.

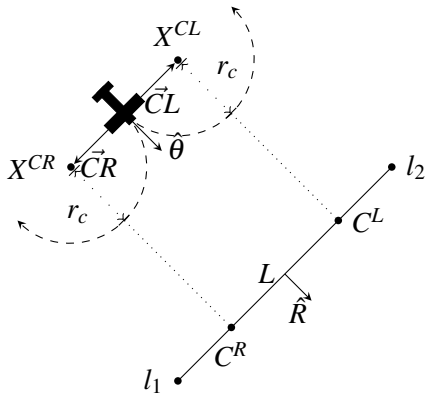


Figure 5.6: Representation of the maneuvers' center points

The points along the avoidance maneuvers which are closest to the wall would be defined as:

$$X^{TL} = X^{CL} + r_c \hat{R} \tag{5.12}$$

$$X^{TR} = X^{CR} + r_c \hat{R} \tag{5.13}$$

if there were no delays. However, since these are present, the time at which the avoidance maneuver is started and the time at which the inviability of the maneuver is detected are different. As such, this fact must be taken into account. We do this by considering, in terms of the avoidance maneuvers, not only the vehicle's minimum turning radius but also a tolerance distance ε, as explained when we introduced the safe avoidance controller (see theorem 1). The value of ε is determined such that it is possible for the vehicle to start the avoidance maneuver in time.

$$X^{TL} = X^{CL} + (r_c + \varepsilon)\hat{R} \tag{5.14}$$

$$X^{TR} = X^{CR} + (r_c + \varepsilon)\hat{R} \tag{5.15}$$

Even though we could compute the position of the centers of the avoidance maneuvers at the instants at which these actually start (i.e. take the delay into account when computing X^{CL} and X^{CR}) and consider the minimum turning radius r_c, by considering an additional tolerance ε, not only we no longer need to compute the centers' future positions, but we are also increasing the system's safety relative to very particular scenarios. One example of such a scenario is shown in figure 5.7.

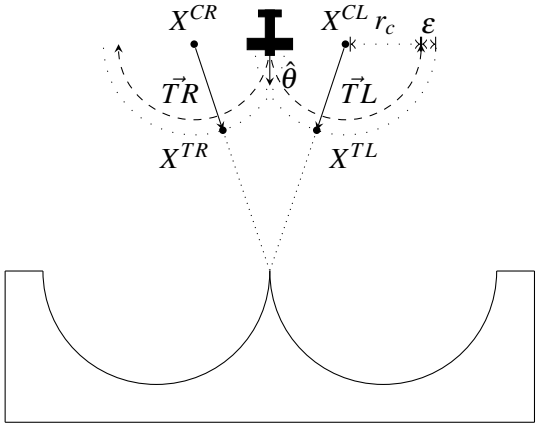


Figure 5.7: Representation of the maneuvers' test points for a particular case

Although this type of obstacles is not considered in this work, we mention them to illustrate the need to compute the test points as before. As we can see in figure 5.7, if we do not consider the additional tolerance ε, the vehicle would enter the obstacle, as it would pass undetected - since the vehicle is aligned with the obstacle, none of the avoidance maneuvers would intersect it and, consequently, the vehicle would hit the obstacle.

After computing the test points, we must check if these are inside the obstacle or not.

Condition 1 (Test condition) *Consider an obstacle described by definition 22 and let X_T denote a given test point. We have that:*

i) If $(\vec{X_T M}^T \hat{R}) > 0$, then X_T is outside the given obstacle;

ii) If $(\vec{X_T M}^T \hat{R}) = 0$, then X_T is on the obstacle's boundary;

iii) If $(\vec{X_T M}^T \hat{R}) < 0$, then X_T is inside the obstacle;

where the vector $\vec{X_T M}$ is defined as $\vec{X_T M} = M - X_T$ and M is the wall's midpoint, $M = \frac{1}{2}(V_0 + V_1)$. Only when i) holds, the avoidance maneuver is viable.

This condition is illustrated in figure 5.8.

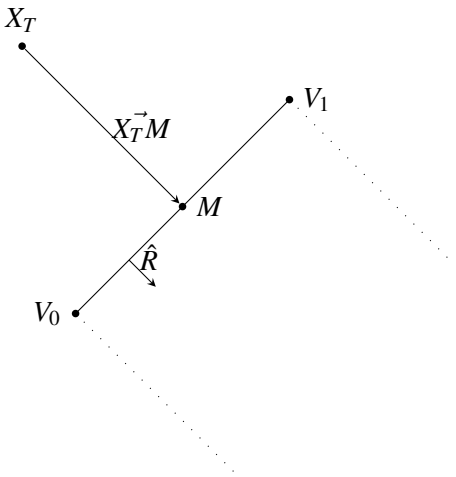


Figure 5.8: Illustration of the point-in-set test

5.1.2 Convex obstacles

Definition 28 (Convex polygon) *A convex polygon is defined by the intersection of a set of shifted half planes, each of which defined as in definition 22.*

An example of a convex polygon is illustrated in figure 5.9.

When dealing with convex polygonal obstacles, there is no need to test every edge of the polygon, as it is impossible for the vehicle, at a given position, to be on a collision course with all of the polygon's edges. So, we use definition 26 to determine with which edges of the polygon the vehicle might collide. This way, only after having determining which edges should be analyzed,

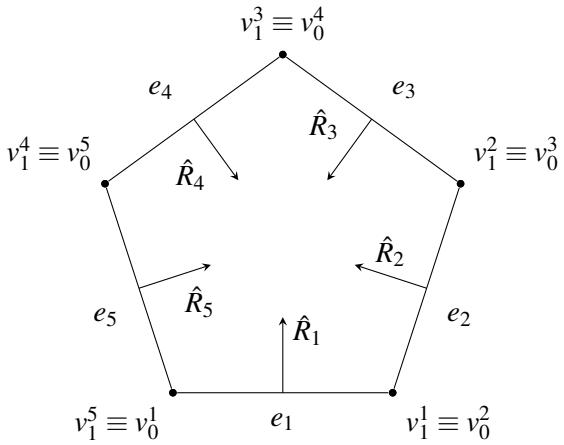


Figure 5.9: Convex polygonal obstacle

should we compute the corresponding test points.
Still, there is a particular situation that we must take into account and that differs from the simple wall. As we can see in figure 5.10, it is clear that the previously defined test points for the simple wall are no longer enough to detect inviable trajectories since the vertices of the polygon might be the closest points and will not be detected by the previous tests.

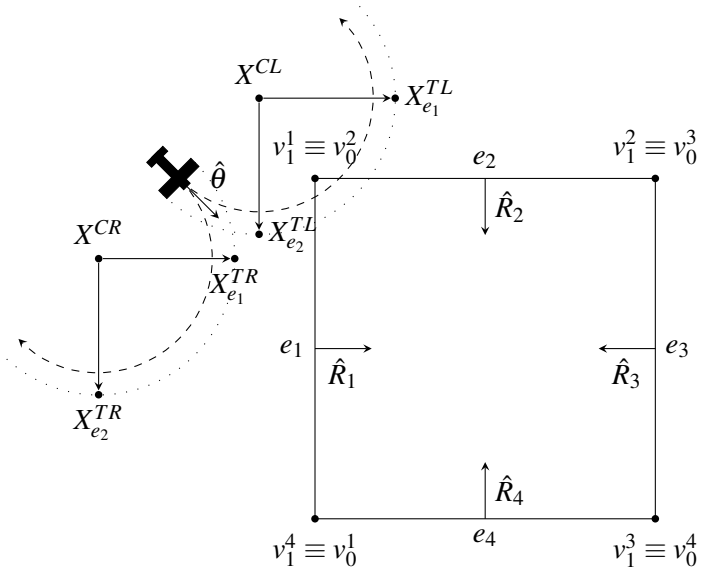


Figure 5.10: Issue occurring from the use of the simple wall approach for convex polygonal obstacles

This way, when we are testing the viability of an avoidance maneuver it is essential to define and analyze:

1. Test points that are closest to the edges, using the point projection onto a straight line.

2. The distance from the center of the maneuver to all the vertices of the polygon.

Condition 2 (Extended test condition for convex polygons) *Consider a N-sided obstacle described by definition 28. Let X_T denote a given test point and $\{d_0^i, d_1^i\}$ the distances between the center of the avoidance maneuver and the vertices V_0 and V_1 of the edge e_i. We have that:*

i) If $\max_i \left\{ \vec{X_T M_i}^T \hat{R}_i \right\} < 0$, *then X_T is inside the given obstacle;*

ii) If $\max_i \left\{ \vec{X_T M_i}^T \hat{R}_i \right\} = 0$, *then X_T is on the obstacle's boundary;*

iii) If $\max_i \left\{ \vec{X_T M_i}^T \hat{R}_i \right\} > 0$, *then X_T is outside the obstacle;*

where the vector $\vec{X_T M_i}$ is defined as $\vec{X_T M_i} = M_i - X_T$ and M_i is the i^{th} wall's midpoint ($M_i = \frac{1}{2}(V_0^i + V_1^i)$). Only when iii) holds, may the avoidance maneuver be viable, as it is also necessary that the distances $\{d_0^i, d_1^i\}$ are greater than $(r_c + \varepsilon)$ for all $i \in \{1, 2, ..., N\}$.

We illustrate our strategy in figure 5.11 where, for simplicity reasons, only one edge is shown instead of the whole polygon. Note that dL_0^i and dL_1^i are the distances between the center of the left avoidance maneuver and the vertices V_0^i and V_1^i, respectively. Similarly, dR_0^i and dR_1^i are the distances between the center of the right avoidance maneuver and the same vertices.

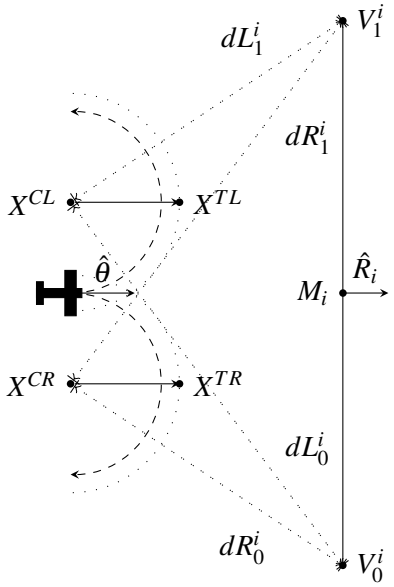


Figure 5.11: Illustration of the test point approach for a single edge

5.1.3 Non-convex obstacles

Having successfully extended our approach to convex polygonal obstacles, it is natural to try and extend it to non-convex obstacles too. One of the first issues of extending the convex approach to non-convex polygons is that the procedure we use to determine with which edges the vehicle might collide fails. At the same time, the procedure we use to find whether or not the determined test points are inside the polygon also fails, so we are forced to review our approach.
We start by rethinking how to determine the edges for which we will compute the test points for a non-convex polygon. The problem with our current approach is that edges that are behind other edges will be unnecessarily considered, so definition 26 is not enough. Thus, we need to devise a condition that excludes every edge that respects definition 26 but is behind some other edge - what we call *invisible edges*- as these do not pose a risk to the vehicle's trajectory. In what follows we devise a method to determine which edges to test, using figure 5.12 as an example.

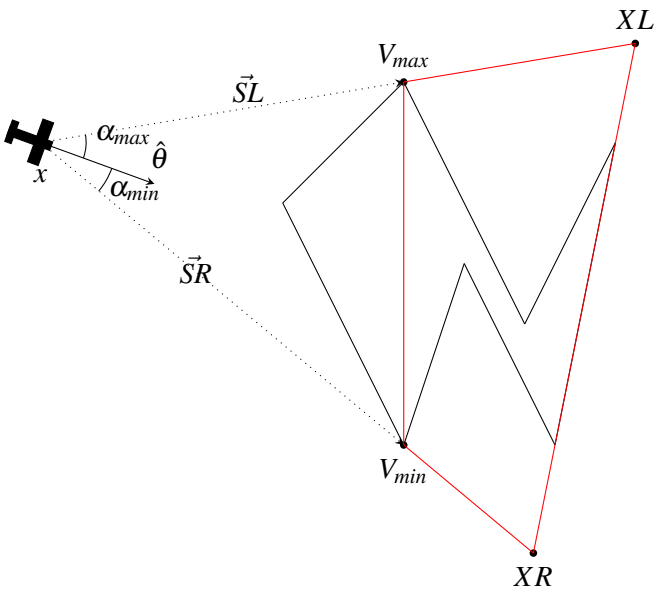


Figure 5.12: Visible and invisible edges of a non-convex polygon

Let α_i be the angle between the orientation of the vehicle, $\hat{\theta}$, and the vector that goes from x to the i^{th} vertex of the polygon. So, given all vertices, we consider the ones which result in the largest (most positive) and smallest (most negative) α.
Let V_{max} and V_{min} denote such vertices and $\vec{SL}$ and $\vec{SR}$ the correspondent vectors.

$$\vec{SL} = V_{max} - x \tag{5.16}$$

$$\vec{SR} = V_{min} - x \tag{5.17}$$

The vectors $\vec{SL}$ and $\vec{SR}$ will be used to form a new polygon (red polygon in figure 5.12), where the

edges with both vertices in it will not be tested since they do not represent any risk of collision. This polygon is formed by the vertices $\{V_{max}, XL, XR, V_{min}\}$. XL can be obtained by adding the vector $\vec{SL}$ multiplied by a scalar $k_1 \in \mathbb{R}$ to the point V_{max}. The same process can be used to determine XR, multiplying $\vec{SR}$. Notice that these scalars have to be large enough so that the new polygon defined by $\{V_{max}, XL, XR, V_{min}\}$ contains all the *invisible edges*.
Now that we have solved the first issue, let us rethink the current solution to the *point in polygon* problem. Looking at figure 5.13, we can see why the method we have devised to determine if a point is in $\mathscr{Q}$ no longer works. Even though condition 2 indicates that the test point X_T is outside the obstacle, we can see that this is false.

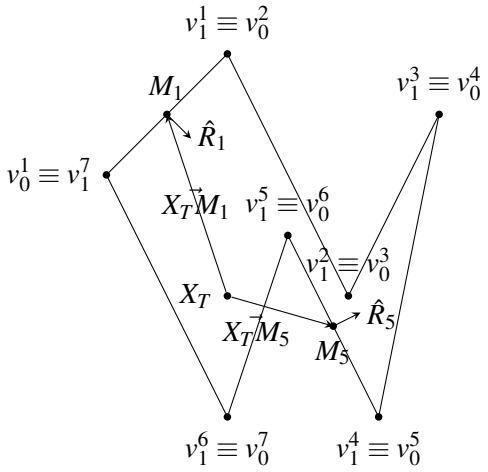


Figure 5.13: Issue occurring from the use of the convex obstacles approach for non-convex obstacles

This being said, how do we deal with non-convexities? One idea is to "convexify" the obstacle by splitting it into several non-overlapping convex obstacles - *partitions*. This way, the "*point in polygon*" problem for a non-convex polygon can be solved by determining whether or not the point is in one of the individual convex obstacles. We can use this procedure to extended the theory used for convex polygon to non-convex polygons by, instead of testing where a test point X_T is relative to the non-convex polygon, testing it relative to the partitions of the non-convex polygon. This way, if the point is in one of the partitions, then it is also in the non-convex polygon.

Remark 1 (Extension of the test condition for non-convex polygonal obstacles) *Assume that there exists a procedure that partitions the non-convex polygon such that each partition is convex. This way, applying condition 2 to each partition, we are able to devise (by extension) a test condition for non-convex polygonal obstacles.*

There are several methods we can use to partition a polygon, however, for the sake of simplicity, we will use triangulation.

5.1.3.1 Triangulation

The following definitions will be useful in understanding the triangulation procedure as well as the underlying concepts.

Definition 29 (Graph) *A graph is a pair $G = (N, E)$ of sets such that the elements of $N = \{n_1, n_2, ..., n_n\}$ are the nodes or vertices of the graph G and the elements $E = \{e_{12}, e_{23}, ..., e_{n-1n}\}$ are its edges (see figure 5.14).*

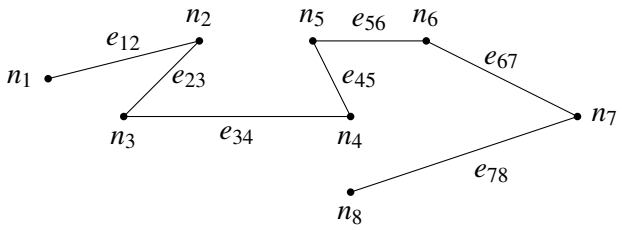


Figure 5.14: Graph

Definition 30 (Adjacent vertices) *Two vertices n_i and n_j of a graph G are adjacent or neighbors if e_{ij} is an edge of G.*

Definition 31 (Path) *A path is a sequence of edges connecting two vertices such that any two adjacent edges in that sequence share a common vertex.*

Definition 32 (Adjacency matrix) *The adjacency matrix A of a graph is defined as follows.*

$$A : a_{ij} = \begin{cases} 1, \textit{ if node i is connected to node j} \\ 0, \textit{ otherwise} \end{cases}$$

If the graph is an undirected graph, i.e. its edges are not directed, the adjacency matrix will be symmetric and if each vertex isn't directly connected to itself, its diagonal will be zero.
Now that we introduced some essential definitions let us return to the triangulation procedure. The triangulation of a polygon is a method that divides the polygon into a set of triangles. To do this we use the *subtracting ears* method. When we refer to an *ear* we are talking about a triangle having two adjacent edges of the polygon as sides, and the remaining side being a polygon's diagonal.

Definition 33 (Diagonal) *A diagonal is a line segment that lies inside a polygon and does not intersect its boundary except at its endpoints.*

Given a polygon, the *subtracting ears* method works as follows.

1. Select a vertex V of the polygon and its two adjacent vertices, $\mathcal{N}(V)$.
2. Verify if the three selected vertices constitute an *ear*, that is, if the line segment connecting V's adjacent vertices is a diagonal of the polygon.
3. If condition 2 is true, remove the *ear* from the polygon and define it as a new partition.
4. If condition 2 is not true, return to condition 1, selecting a new vertex V.
5. Repeat the process until the remaining polygon is a triangle ($dim(A) = 3$).

Since our goal is to partition the obstacle into convex polygons, we modify the *subtracting ears* method so that we stop when the remaining polygon is already convex. Considering the polygon in figure 5.15 as an example, let us apply the method.

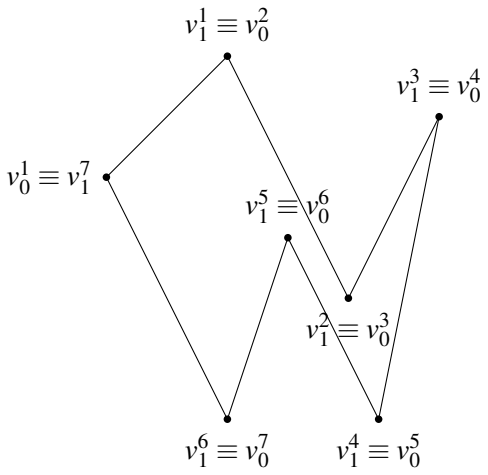


Figure 5.15: Non-convex polygon that will be submited to the triangulation method

Let $\mathcal{V}$ be the set of all vertices of the polygon.

$$\mathcal{V} = \{v_0^1, v_0^2, v_0^3, v_0^4, v_0^5, v_0^6, v_0^7\}$$

Interpreting the original polygon as an undirected graph $G = (N, E)$ where the nodes are the vertices of the polygon and the edges are the polygon's edges, by computing the graph's adjacency matrix A, given any vertex, we can easily find its adjacent vertices.
Take, for example, the graph representation of the above polygon in figure 5.16.

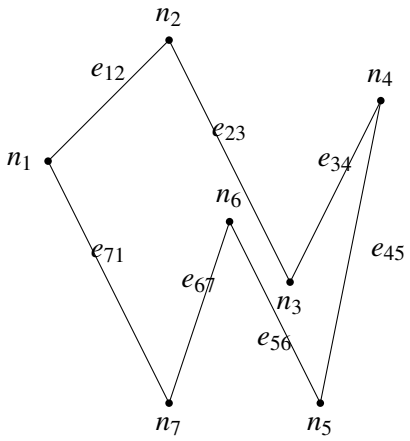


Figure 5.16: Graph representation of the polygon in figure 5.15

We want to know which are the adjacent vertices to the node n_1, the initially selected vertex. To do this, we compute the graph's adjacency matrix, which yields:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Thus, by looking at the first row, we see that node n_1 is connected to nodes n_2 and n_7 - its adjacent vertices. Next, we need to determine if the segment formed by nodes n_2 and n_7 is a diagonal of the polygon. This segment, D_{27}, represented in figure 5.17, is a diagonal if and only if definition 33 holds.

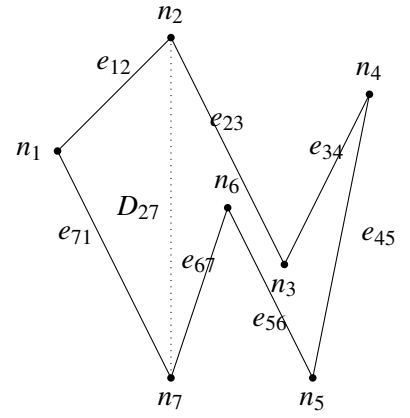


Figure 5.17: Illustration of a diagonal candidate

In this case, we conclude that D_{27} is a diagonal since it does not intersect the polygon at any point(s) other than at its vertices and it is completely inside it.
The process of removing the *ear* from the polygon consists of defining two new polygons - the new partition and the remaining polygon. It will be on the remaining polygon that we will continue to perform triangulation until it is convex.
Before continuing, we must explain how to define the partition. To do this, we characterize the triangle defined by the selected node and its adjacent nodes. Consider figure 5.18 for example, and let the vector $\vec{v} = [v_x, v_y] = n_2 - n_7$ be the vector that goes from node n_7 to node n_2.

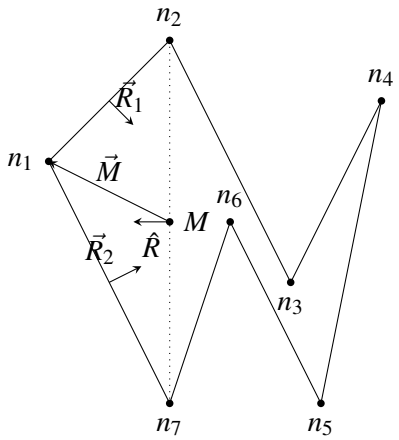


Figure 5.18: Illustration of an *ear* and the remaining polygon

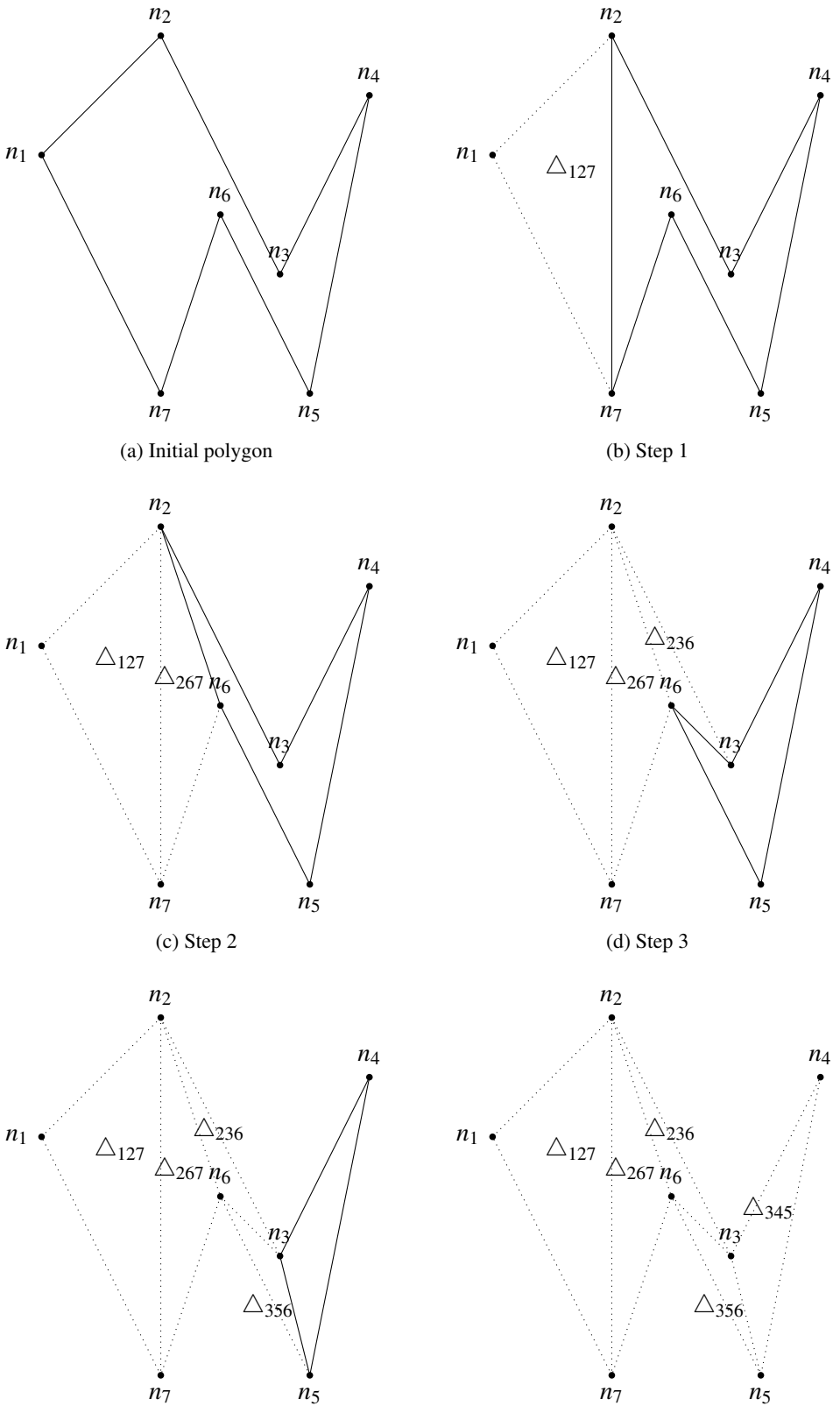
The candidate to normal vector of the diagonal D_{27} is :

$$\hat{R}_c = \frac{[-v_y, v_x]}{\|\vec{v}\|} \tag{5.18}$$

Let M be D_{27}'s midpoint and $\vec{M}$ the vector that points from M to node n_1. The normal vector $\hat{R}$, in the diagonal D_{27}, should be pointing to the interior of the triangle. So, using the inner product between $\vec{M}$ and $\vec{R}_c$ we know that if:

$$\begin{cases} \vec{M}^T \hat{R}_c > 0 \text{ ,then } \hat{R} = \hat{R}_c \\ \vec{M}^T \hat{R}_c < 0 \text{ ,then } \hat{R} = -\hat{R}_c \end{cases}$$

With the three edges completely defined $e_{12} = \left\{n_1, n_2, \vec{R}_1\right\}$, $e_{17} = \left\{n_1, n_7, \vec{R}_2\right\}$ and $e_{27} = \left\{n_2, n_7, \vec{R}\right\}$, we can now define the triangle $\triangle_{127}$, that is the new partition.
The remaining polygon excludes the edges between the selected node and the adjacent nodes and adds a new edge defined by the two adjacent nodes. A complete example of the triangulation procedure is shown in figure 5.19.



(a) Initial polygon (b) Step 1

(c) Step 2 (d) Step 3

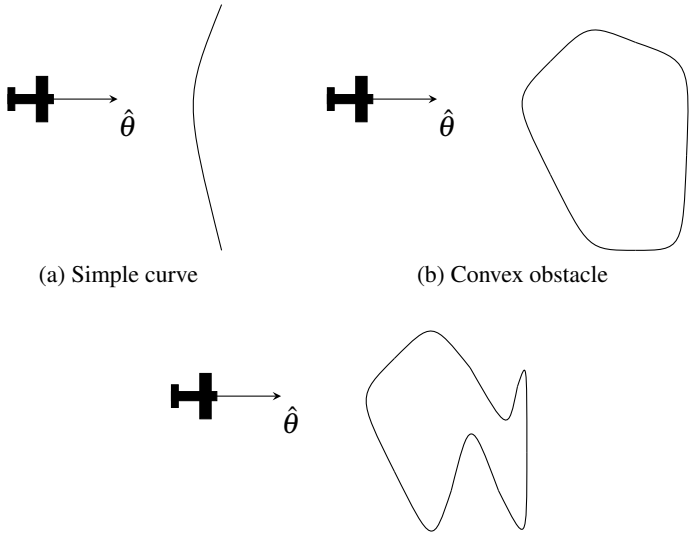
(e) Step 4 (f) Step 5

Figure 5.19: Triangulation method

Having partioned the initial non-convex obstacle into a set of N convex partitions, we can now test if the avoidance maneuvers are viable, i.e., if the test points are in the obstacle as explained in remark 1. Notice that the distances from the center of the avoidance maneuver to the vertices must be considered too.

5.2 Cubic spline-based obstacles

As we have done with the polygonal obstacles, where we began by studying a simple scenario that we would later extend to more complex problems, here we will also start by analyzing a simple curve (figure 5.20(a) defined by a cubic spline and then extend our results to the case where we have a convex obstacle (figure 5.20(b) defined by a given number of cubic splines. Finally we let this obstacle be non-convex (figure 5.20(c).



(a) Simple curve (b) Convex obstacle

(c) Non-convex obstacle

Figure 5.20: Different spline-based obstacles

5.2.1 Simple curve

When we studied the shifted half plane problem in the previous section, we developed an approach based on test points. This approach took advantage of the fact that, for a shifted half plane, there is only one normal direction, so it is only necessary to test one point instead of solving the point projection problem. However, given a curve this is no longer true, so the approach based on test points becomes inviable. Thus, for these type of obstacles we will have to solve the point projection problem, in order to implement the distance function used in the proof of theorem 1 for the state transition conditions. Note that in the approach we presented for polygonal sets we avoided an explicit implementation of the distance to a set function presented in that proof by using the concept of test point.
We start by defining the tools we will use to describe our obstacle.

Definition 34 (Simple curve) *The boundary of a curve segment is described by a cubic spline, which is completely defined by:*

$$S = \{x \in \mathbb{R}^2 : x = KS_0(s), s \in [s_-, s_+]\} \tag{5.19}$$

where $K = \begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \end{bmatrix}$ *and* $S_0(s) = [s^3\ s^2\ s\ 1]^T$. *For the particular case of a infinite simple curve, we have that* s_- *and* s_+ *are equal to* $-\infty$ *and* $+\infty$, *respectively.*

Let P be a generic point for which we want to find the minimum distance to the simple curve (see figure 5.21).

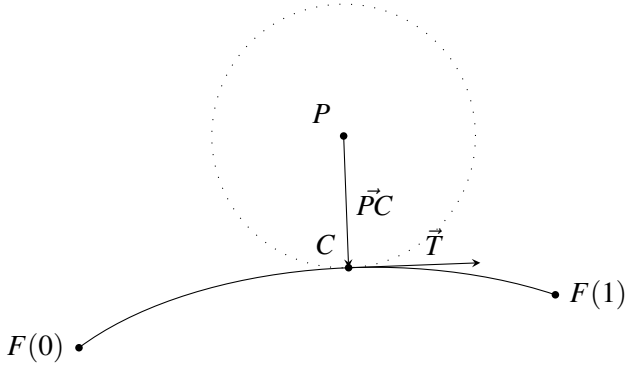


Figure 5.21: Point projection problem for a simple curve

The square of the distance between P and a generic point on the curve, $[x(s), y(s)]^T$ is given by:

$$D(s) = (x(s) - x_0)^2 + (y(s) - y_0)^2 \tag{5.20}$$

The optimal parameter s^* that minimizes $D(s)$ determines $C = [x(s^*), y(s^*)]^T$. The vector $\vec{PC}$ is perpendicular to the tangent vector, $\vec{T}$, at C. The minimum distance between P and the curve is given by the norm of the vector $\vec{PC}$.

Conventional optimization techniques such as quadratic minimization or Newton's method can be used to solve this problem. However, and based on [25], none of them are sufficiently good for real time simulations, since quadratic minimization presents a slow rate of convergence and Newton's method can result in large errors if the initial estimate is far away from the optimal value. Still, and as is exposed in [25], the convergence properties of both methods complement each other in terms of their weaknesses. Quadratic minimization is good at improving coarse estimates while Newton's method is good at converging to the optimal value if the initial estimates are sufficiently good. As such, we will use both methods, combining their advantages, resulting in a better solution for the problem of finding the closest point on a curve, that is, finding $[x(s), y(s)]^T$ such that s minimizes $D(s)$.

5.2.1.1 Quadratic minimization

Quadratic minimization uses interpolation to obtain a quadratic polynomial that approximates $D(s)$ at the initial estimates. To determine these initial estimates we take into consideration the parameter of the spline $F(s)$, $s_- < s < s_+$ and we usually set the initial estimates as follows.

$$\tilde{s_1} = s_- \tag{5.21}$$

$$\tilde{s_2} = s_- + \frac{(s_+ - s_-)}{2} \tag{5.22}$$

$$\tilde{s_3} = s_+ \tag{5.23}$$

$$\tag{5.24}$$

The quadratic polynomial that interpolates $D(s)$ at $\tilde{s}_1$, $\tilde{s}_2$ and $\tilde{s}_3$ is given by,

$$P(s) = \frac{(s-\tilde{s}_2)(s-\tilde{s}_3)}{(\tilde{s}_1-\tilde{s}_2)(\tilde{s}_1-\tilde{s}_3)}D(\tilde{s}_1) + \frac{(s-\tilde{s}_1)(s-\tilde{s}_3)}{(\tilde{s}_2-\tilde{s}_1)(\tilde{s}_2-\tilde{s}_3)}D(\tilde{s}_2) + \frac{(s-\tilde{s}_1)(s-\tilde{s}_2)}{(\tilde{s}_3-\tilde{s}_1)(\tilde{s}_3-\tilde{s}_2)}D(\tilde{s}_3) \tag{5.25}$$

and the parameter s^* that minimizes $P(s)$ is, for the m^{th} iteration:

$$s^{*,m} = \frac{1}{2}\frac{(\tilde{s}_2^2-\tilde{s}_3^2)D(\tilde{s}_1) + (\tilde{s}_3^2-\tilde{s}_1^2)D(\tilde{s}_2) + (\tilde{s}_1^2-\tilde{s}_2^2)D(\tilde{s}_3)}{(\tilde{s}_2-\tilde{s}_3)D(\tilde{s}_1) + (\tilde{s}_3-\tilde{s}_1)D(\tilde{s}_2) + (\tilde{s}_1-\tilde{s}_2)D(\tilde{s}_3)}, m = 1,2,3,... \tag{5.26}$$

So the method consists in determining $s^{*,m}$ using $\tilde{s}_1$, $\tilde{s}_2$ and $\tilde{s}_3$, and eliminating the estimate that corresponds to the largest $P(s)$. However, it should be noticed that, in some cases, the estimate converges to a value outside the range of s. This usually happens when the optimal value s^* is near the boundaries of the spline. To solve this, we force the estimate to be equal to one of the limits of s, depending on the closest value.
Another important remark is that, when the new estimate $s^{*,m}$ is already in the set of initial estimates, $\{\tilde{s}_1, \tilde{s}_2, \tilde{s}_3\}$, the method should stop, in order to prevent singularities in $P(s)$. The method is presented in algorithm 1.

Algorithm 1 Quadratic Minimization Method

1: *Let* $\tilde{s}_1 = s_-$, $\tilde{s}_2 = s_- + \frac{(s_+-s_-)}{2}$, and $\tilde{s}_3 = s_+$
2: **repeat**
3: $s^{*,m} = f(\tilde{s}_1, \tilde{s}_2, \tilde{s}_3)$ (equation 5.26)
4: Remove argmax $P(s)$, $s \in \{\tilde{s}_1, \tilde{s}_2, \tilde{s}_3, s^{*,m}\}$
5: **until** $m = 10$ or $s^{*,m} \in \{\tilde{s}_1, \tilde{s}_2, \tilde{s}_3\}$

Based on our results for different examples where we used the quadratic minimization method to find the optimal parameter s^* that minimizes $D(s)$, we concluded that ten iterations are usually enough to improve the initial estimates so that these can be used successfully by Newton's method.

5.2.1.2 Newton's method

Newton's method is used for solving equations like $f(x) = 0$, starting with an initial estimate x_0 and generating a succession $\{x_n\}$ in a recurrent way. Each new estimate, x_{n+1}, is given by the abscissa of the intersection between the tangent line in the point $(x_n, f(x_n))$ and the x axis (see figure 5.22).

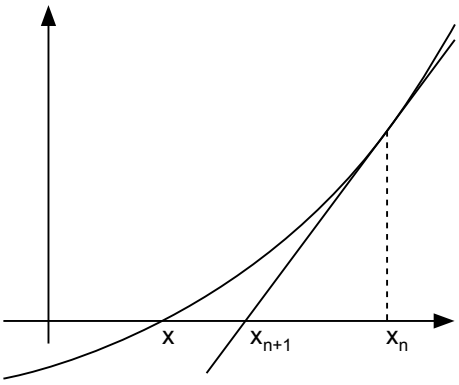


Figure 5.22: Newton's method

So the recurrence expression of x_{n+1} in order to x_n is easily obtained if we determine the equation that describes the tangent line in $(x_n, f(x_n))$ ([26]).

$$y = f(x_n) + f'(x_n)(x_{n+1} - x_n) \tag{5.27}$$

Because of what we have said before, this equation will pass through the point $(x(n+1), 0)$ and so the recurrent equation becomes,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{5.28}$$

Newton's method is presented in algorithm 2.

Algorithm 2 Newton's Method

1: *Let* $x(0) = x_0$
2: **repeat**
3: $\quad x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$
4: **until** $\|x_{n+1} - x_n\| \leq \delta$

However, there are particular situations where Newton's method does not converge to the solution, as in figure 5.23.

In this case, the function $f(x)$ has a point with a null derivative, i.e., f'(x) = 0, and as we can see, an unstable behavior takes place when Newton's method tries to find the solution. A similar case

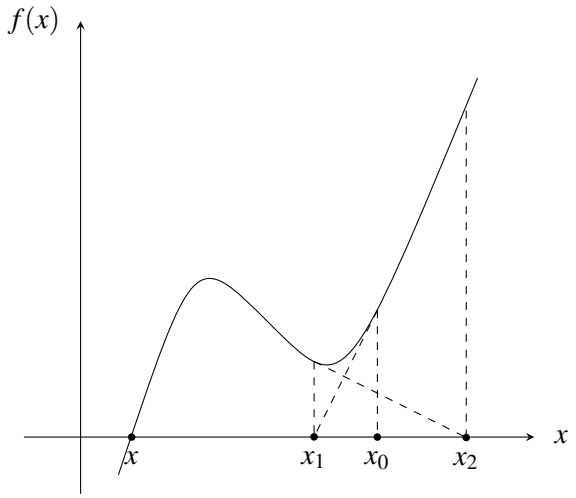


Figure 5.23: Newton's method failure

is presented in figure 5.24 where the succession does not converge to the solution due to a change in convexity.

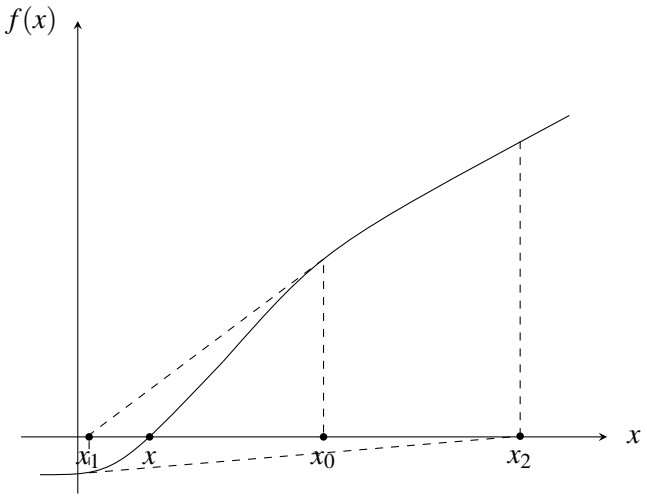


Figure 5.24: Newton's method failure (2)

Theorem 2 (Sufficient conditions for the convergence of Newton's method) *Let $f \in C^2([a,b];\mathbb{R})$ such that $f'(x) \neq 0$ and $f''(x) \leq 0$ or $f''(x) \geq 0 \in [a,b]$ and s the only zero of f in [a,b]. If $f(x_0)f''(x_0) \geq 0$ holds for an initial point $x_0 \in [a,b]$ then the succession produced by Newton's method converges to s.*

The above theorem gives sufficient but not necessary conditions to the convergence of the method,

as there are cases where they do not hold and the method converges correctly[1].
When we have minimization or maximization problems, Newton's method can be used to find local minima or maxima of objective functions. In our case, this function is the distance function $D(s) = (x(s) - x_0)^2 + (y(s) - y_0)^2$ and we want to find its minimizer, i.e, the value s^* that satisfies $D'(s^*) = 0$. So, instead of applying Newton's method to the function $D(s)$ we will apply it to its derivative, $D'(s)$, and the recurrent equation becomes,

$$s^{*,n+1} = s^{*,n} - \frac{D'(s^{*,n})}{D''(s^{*,n})}, n = 0, 1, 2, 3, ... \tag{5.29}$$

Returning to our problem, we use both methods, applying quadratic minimization first, followed by Newton's method, to determine which points of the simple curve are closest to the centers of the avoidance maneuvers (see figure 5.25). With these points, we can easily obtain the distance between the centers of the avoidance maneuvers to the obstacle, and determine if the maneuvers are viable or not. It should be noticed, however, that our method may fail if the derivative of the distance function, $D'(s)$, presents a behavior as the ones described before when Newton's method does not converge to the solution.

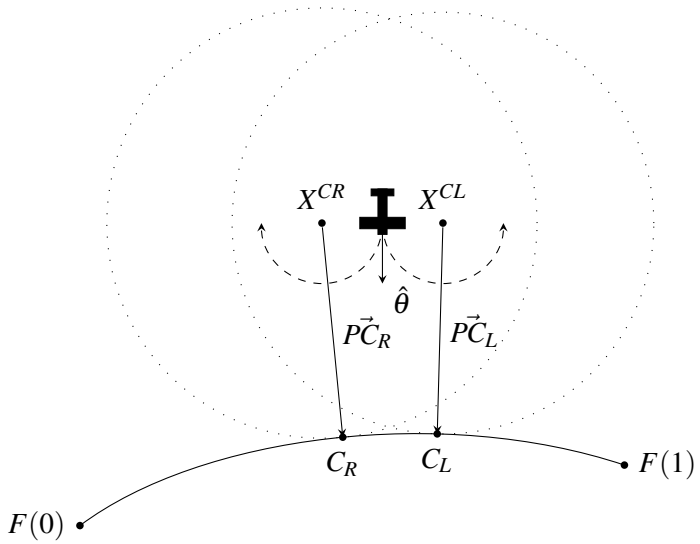


Figure 5.25: Closest points on the curve to the centers of the avoidance maneuvers

Condition 3 (Test condition) *Consider an obstacle described by definition 34. We have that:*

i) If $d_S(X^{CL}) > (r_c + \varepsilon)$, then the left avoidance maneuver is viable;

ii) If $d_S(X^{CR}) > (r_c + \varepsilon)$, then the right avoidance maneuver is viable;

[1]The proof for theorem 2, as well as examples where the method fails to converge, can be found in ([26])

where $d_S(X^{CL})$ and $d_S(X^{CR})$ represent the distance between the centers of the left and right avoidance maneuvers, respectively, to the spline S of the obstacle.

5.2.2 Convex obstacles

Definition 35 (Spline-based convex set) *A spline-based convex set is defined by a set of several segments, each one characterized by $\{K, x_0, x_1\}$ such that:*

$$\{x \in \mathbb{R}^2 : x = KS(s), KS(0) = x_0, KS(1) = x_1, s \in [0,1]\}$$

We assume that the convex set, $\{S_1 ... S_N\}$ is such that S_i connects x_i to x_{i+1} for $i \in \{1,...,N-1\}$ and S_N connects x_N to x_1. We also assume that this set is such that S_i is $C^2[0,1]$ for $i \in \{1,...,N\}$.

An example of a spline-based convex set is shown in figure 5.26.

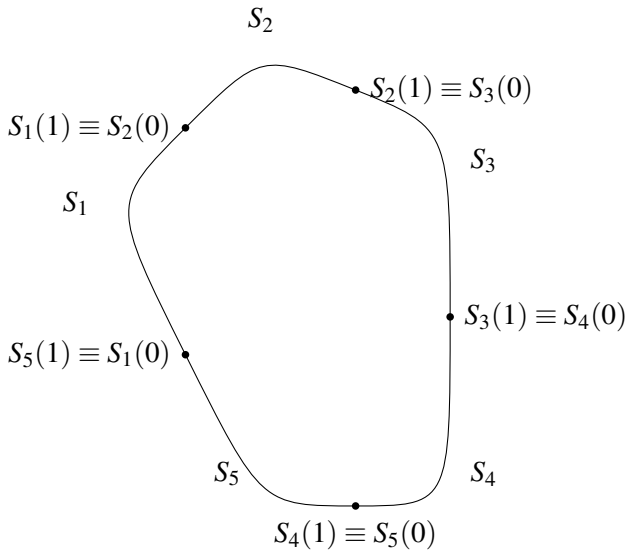


Figure 5.26: Spline-based convex set

Since we now have more than one spline, the way to determine the viability of an avoidance maneuver will differ from the previous scenario where we had only one spline - we solve the problem stated in the previous section for all the splines in the convex set. Having computed all the distances between the centers of the avoidance maneuvers and the spline S_i, we take the minimum as the distance to the obstacle.

Condition 4 (Extended test condition for convex spline-based obstacles) *Consider an obstacle described as in definition 35. We have that:*

i) If $\min_i \left\{ d_{S_i}(X^{CL}) \right\} > (r_c + \varepsilon)$*, then the left avoidance maneuver is viable;*

ii) If $\min_i \left\{ d_{S_i}(X^{CR}) \right\} > (r_c + \varepsilon)$*, then the right avoidance maneuver is viable;*

where $d_{S_i}(X^{CL})$ *and* $d_{S_i}(X^{CR})$ *represent the distance between the centers of the left and right avoidance maneuvers, respectively, to the spline* S_i *of the obstacle.*

5.2.3 Non-convex obstacles

Definition 36 (Spline-based non-convex set) *A spline-based non-convex set is defined the same way as in definition 35, however, the set is allowed to have non-convexities.*

An example of a spline-based non-convex set is shown in figure 5.27.

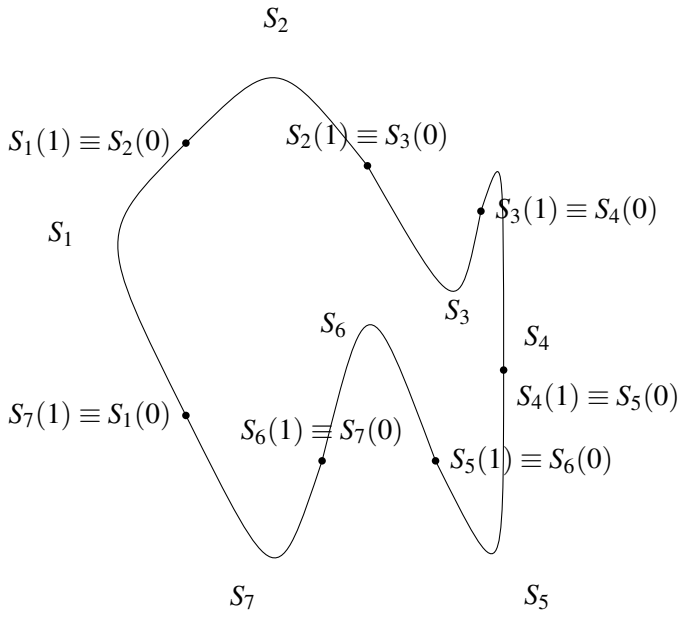
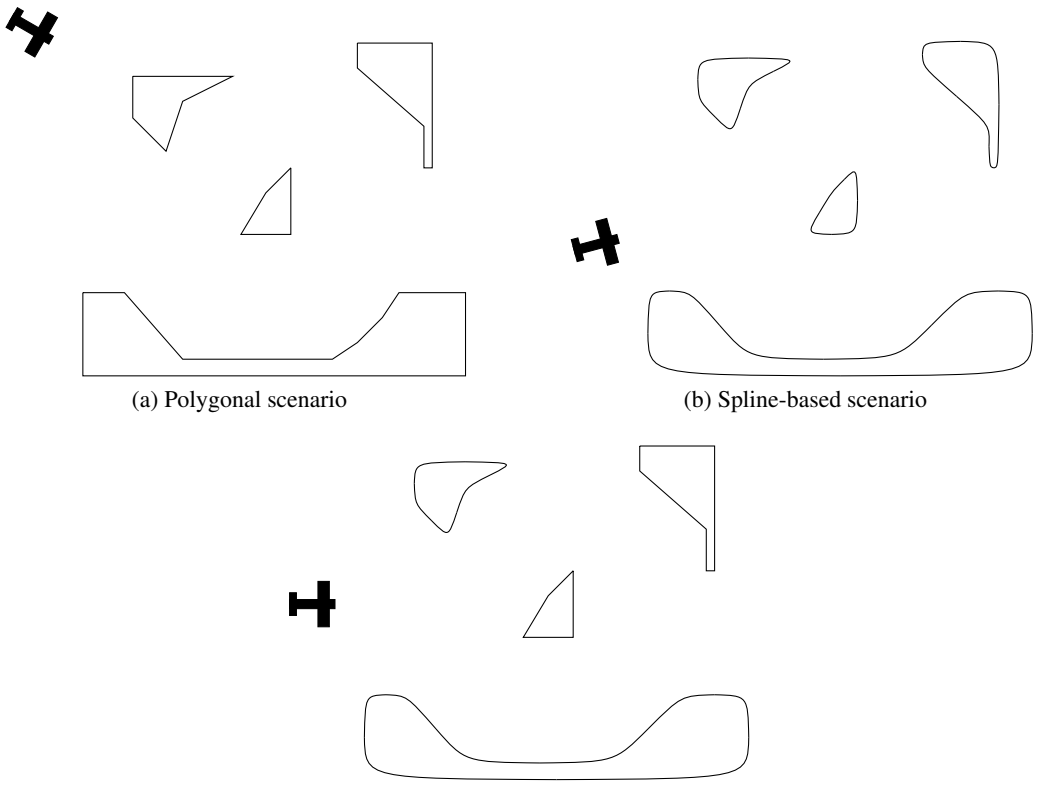


Figure 5.27: Spline-based non-convex set

The approach for this type of sets is the same as for the spline-based convex set as the distance function is the same regardless of convexity.

5.3 Multiple obstacles

In this section we will extend our avoidance strategy from a single obstacle to a multiple obstacles scenario. Our approach is valid for different situations, such as the polygonal scenarios shown in figure 5.28(a), or spline-based scenarios in figure 5.28(b), or even for scenarios where both types of obstacles are present, as presented in figure 5.28(c).



(a) Polygonal scenario

(b) Spline-based scenario

(c) Polygonal and spline-based scenario

Figure 5.28: Multiple obstacles scenario

We extend our approach to multiple obstacles by applying the previously developed avoidance strategies to each obstacle individually, analyzing the viability of the correspondent avoidance maneuvers. However, in order to consider the obstacles separately, we must impose a minimum distance between all the obstacles. This distance should be greater than a minimum safe distance that guarantees that all the avoidance maneuvers for each obstacle, regardless of the vehicle's orientation, are executed without colliding with other obstacles.
In other words, we have to ensure that the avoidance trajectories remain viable even though we are adding more obstacles to our problem.

Theorem 3 *Consider the unicycle model described by system 3.22, controlled by the hybrid automaton shown in figure 5.4 and a set of obstacles* $\{\mathscr{Q}_1, \mathscr{Q}_2, ..., \mathscr{Q}_N\}$. *Assuming that all the avoid-*

ance maneuvers departing from the vehicle's initial state $x(t_0) = x_0$ are viable and the minimum distance between any two obstacles is greater than or equal to a certain distance $d_{\mathscr{Q}}$, then the following property holds:

$$x \notin \mathscr{Q}_i \, , \, \forall t > t_0 \, , \, \forall i \in \{1, 2, ..., N\}$$

In other words, theorem 1 holds for all obstacles.

Proof. By requiring a minimum distance $d_{\mathscr{Q}}$ between all obstacles, we are assuring that our approach can be applied to each obstacle individually, without the problem of any other obstacle affecting the avoidance maneuvers of the vehicle. ■

Chapter 6

A partially known world

In this chapter we will study how to deal with a partially or completely unknown environment. To do this, we will find how to dynamically create over approximations of the real obstacles based on what the vehicle observes at each moment. Notice, however, that we will only cover observability for polygonal obstacles, and not spline-based ones. Also, we will assume that the vehicle has omnidirectional sensing capabilities, with a range greater than or equal to twice the vehicle's minimum turning radius, r_c.
We begin by studying how to determine the observed obstacle from the vehicle's point of view and how to use different observations to update the vehicle's surrounding environment. Then, similarly to what we did in the previous chapter, we proceed by studying convex and non-convex obstacles. Finally we extend our approach to a multiple obstacles scenario.

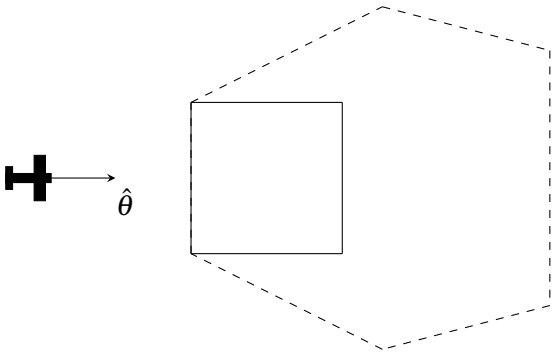


Figure 6.1: Real and observed obstacles (full and dashed, respectively)

6.1 Convex obstacles

As we have mentioned earlier in this chapter, our goal is to find a way to determine what the vehicle is observing at each moment, so that we can relate this with previous information in order

to update the vehicle's knowledge of the surrounding environment - every time new information is available, we are able to improve the vehicle's world model.
One way to determine the observed polygon is to use the *shadow area* method. We choose to use this method as it is simple to implement and sufficient for our purposes. This method allows us to determine the shadow cast by a polygon due the presence of a point-like light emitter. It works by determining which edges are visible (from the emitter's point of view) and using these to construct the shadow. Since, due to implementation reasons, shadows cannot be infinite, the method will also use non-visible edges to construct part of the *shadow polygon*. If we think of the vehicle as the light emitter, it is straightforward to see that the *shadow polygon* will correspondent to the observed polygon.
The first step of this method is to determine which edges of the polygon are visible and which are not. When working with convex polygons, this computation is simple. Take, for example, figure 6.2. Consider edge e_i, the correspondent midpoint M_i and its normal vector $\hat{R}_i$.

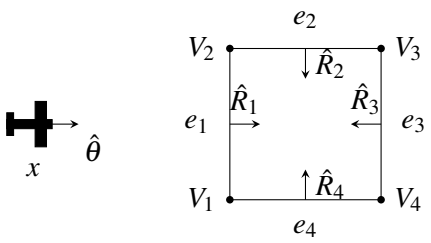


Figure 6.2: Visible edge

It is clear that if $x\vec{M}_i^T \hat{R}_i >= 0$, where the vector $x\vec{M}_i$ is defined as $x\vec{M}_i = M_i - x$, then edge e_i is visible, otherwise it is invisible.
Having determining which edges are visible, we can proceed with the *shadow area* method, whose next step consists of analyzing each edge together with its adjacent edges in order to determine which edges cast shadow - the ones that will be copied to the *shadow polygon*. Invisible edges whose adjacent edges are also invisible will be modified and used to construct what is called the *shadow polygon's back cap*. The remaining edges will not be part of the *shadow polygon*. The algorithm for the *shadow area* method is presented in algorithm 3.

In algorithm 3, an *extrusion* distance is mentioned. This is a finite distance, that defines how much some edges will be moved (extruded) to form the *back cap of the shadow polygon*, as it is not computationally feasible to extrude them infinitely, even though it would be correct. The *extrusion* distance will thus be an important parameter and should be set carefully. A minimum bound for this distance is the maximum distance between the vertices of the polygon for which the *shadow polygon* is to be computed. If this distance is set this way, we can guarantee that the *shadow polygon* contains the real obstacle. To better understand this method, let us apply it to the obstacle in figure 6.2. Let $\mathcal{V} = \{V_1, V_2, V_3, V_4\}$ be the set of all vertices of the polygon and

Algorithm 3 *Shadow Area* Method

Require: Determine which edges of the polygon are visible and which are not
1: Copy all the visible edges to the *shadow polygon* as they are
2: **for** Each edge e of the polygon **do**
3: **for** Each adjacent edge e_n of the edge e **do**
4: **if** e and e_n are both visible **then**
5: Copy e to the *shadow polygon* as it is
6: **else if** Only one of the edges, e or e_n is visible **then**
7: Take the vertex i between the edges e and e_n and move it in the desired *extrusion* distance along the vector that points from the vehicle to the vertex i, forming a new edge
8: **else if** e and e_n are both invisible **then**
9: Take the vertices of the edge e to the shadow area, moving them in the desired *extrusion* distance along the vector that points from the vehicle to the respective vertices, forming a new edge - *back cap of the shadow polygon*.
10: **end if**
11: **end for**
12: **end for**
13: Remove repeated edges of the shadow polygon

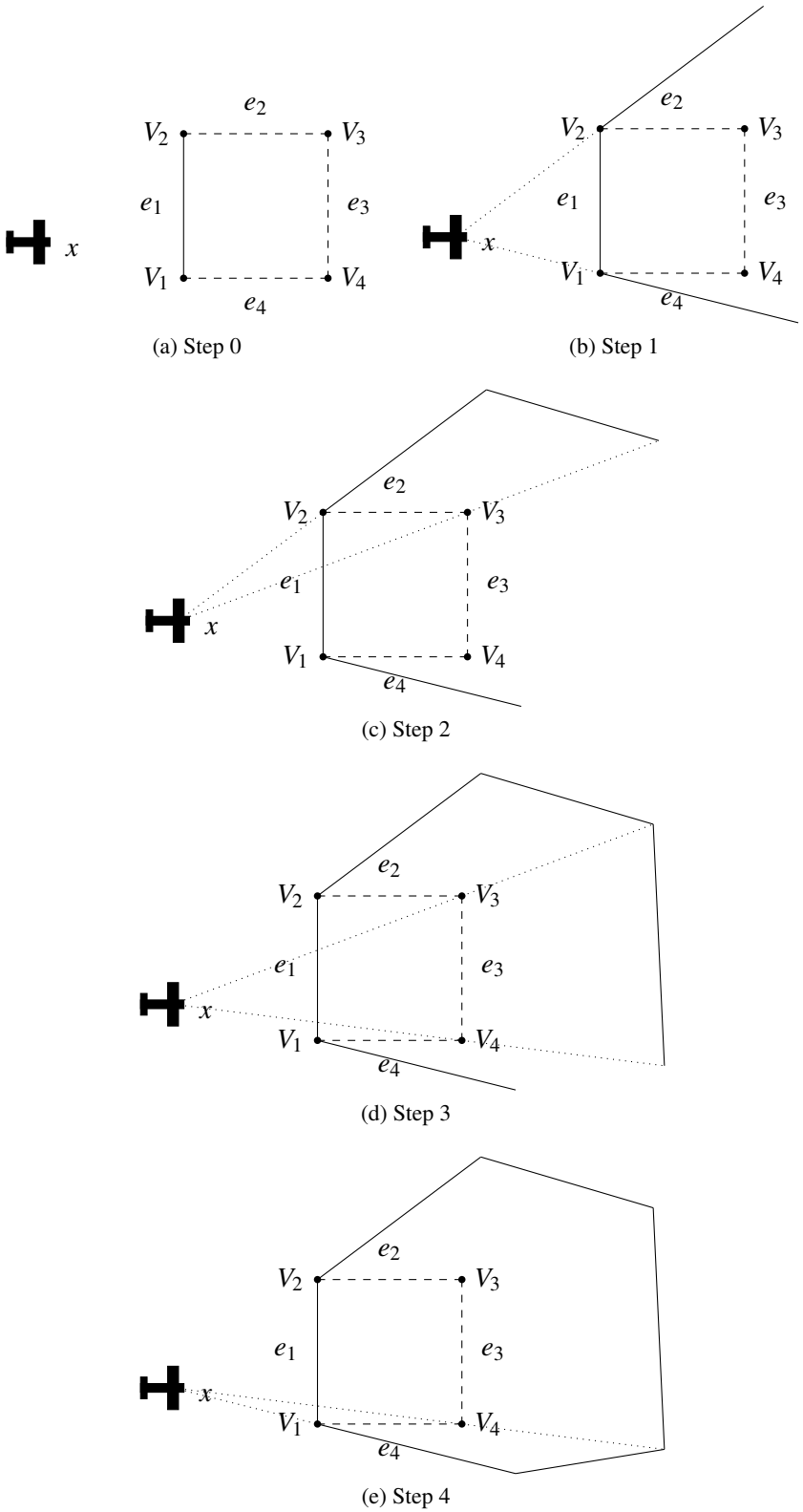
$E = \{e_1, e_2, e_3, e_4\}$ the correspondent edges. According to the vehicle's position we know that only edge e_1 is visible. In figure 6.3, we present a step by step application of algorithm 3. Note that the approximating set is represented by the line in full and it can be seen in its totality in figure 6.3(e).

Now that we have presented a way to determine what the vehicle observes at each moment, we still need to determine a procedure for updating the vehicle's knowledge of the obstacle as it travels. Take, for example, the situation in figure 6.4, where the first observation takes place from the vehicle's initial position, x_1, and the next one from x_2. Note that the time step between x_1 and x_2 is exagerated for illustration purposes.

As we can see in the figure 6.4, different positions yield different observations. If we intersect both observed sets we will obtain a more accurate approximation of the obstacle at x_2, as our knowledge is not limited to what the vehicle sees at that moment but to what has been observed up until that moment (the observation from x_1 included). For this reason we should distinguish between the observed obstacle at instant i, $\hat{Obst}_i$ and the previously known obstacle, $Obst_{i-1}$. The current knowledge of the obstacle will reflect both, as we need to update $Obst_{i-1}$ with $\hat{Obst}_i$, forming the known obstacle at instant i:

$$Obst_i = \hat{Obst}_i \cap Obst_{i-1} \tag{6.1}$$

It is the approximating obstacle $Obst_i$ that must be avoided by the avoidance controller. The only



(a) Step 0

(b) Step 1

(c) Step 2

(d) Step 3

(e) Step 4

Figure 6.3: *Shadow area* method

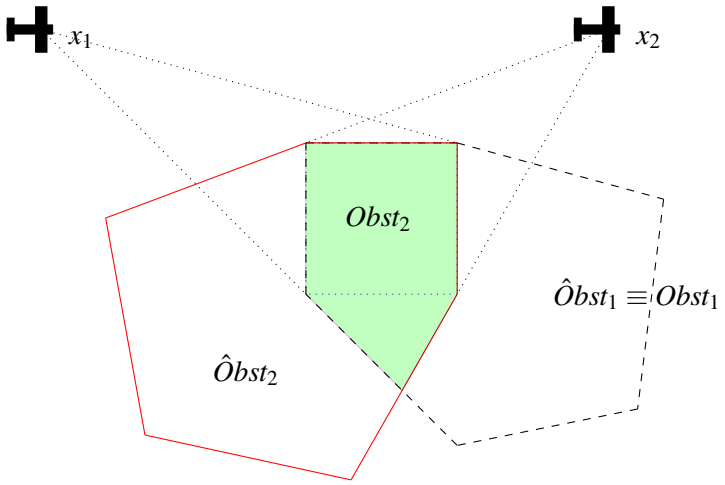


Figure 6.4: Updating procedure

exception is at the first moment since there is no previously available information, so $Obst_0 = \hat{O}bst_0$.

6.2 Non-convex obstacles

The observability problem becomes more complex for non-convex polygons than for convex ones as we can no longer check the visibility of an edge using the same procedure as with the convex polygons. Consider figure 6.5, for example. Analyzing edge e_2 with the same procedure as before, it would be considered a visible edge. In fact, relative to the position of the vehicle, the edge is visible, but only partially, as edges e_1 and e_3 occlude part of it.

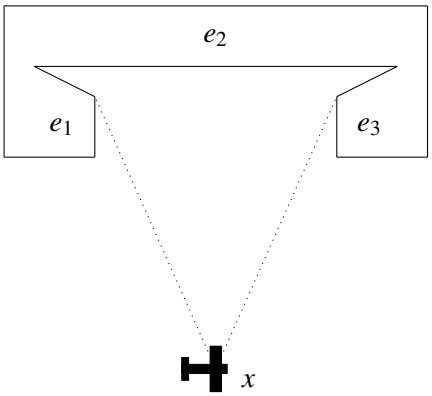


Figure 6.5: Visibility problem for non-convex polygons

In order to solve this problem we recall a method that we have used to find if a test point was

inside a non-convex set - the partitioning procedure which partitioned the set into convex partitions. Then, instead of checking if a given point was inside the non-convex set, we would test if the same point was inside any of the partitions. Here we will use the same method, applied to the observability problem. Since the partitions are convex, the visibility of the edges is no longer a problem. Note, however, that since we are considering individual partitions instead of the whole obstacle, the visibility of an edge may be virtual, as the edge may not exist in the original polygon (i.e. it may be a diagonal of the real polygon).
To illustrate our strategy, recall figure 6.5. As we can see from figure 6.6, the union of the individual *shadow polygons* of the three partitions P_1, P_2 and P_3, results in a valid *shadow polygon* of the non-convex obstacle, represented by the (red) dashed line.

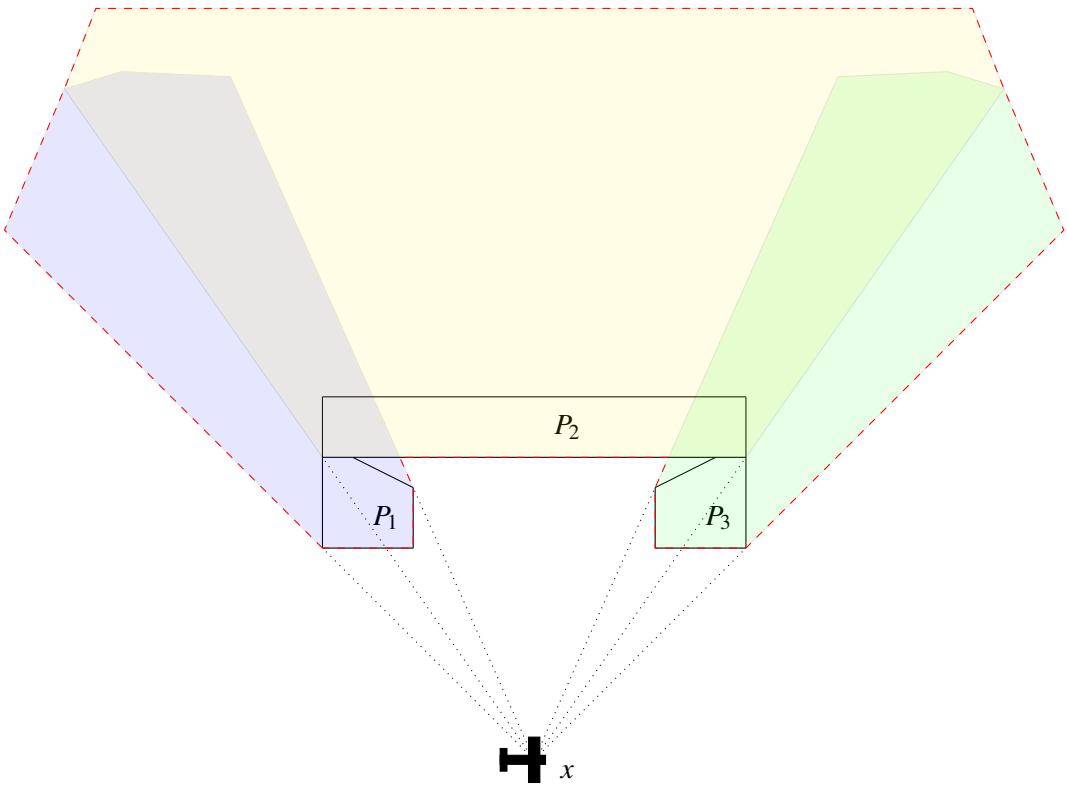


Figure 6.6: Partitions' *shadow polygons*

As with the convex obstacles, here too we progressively update our knowledge of the real obstacle by intersecting the observed obstacle at instant i, $\hat{Obst}_i$, and the previously known obstacle, $Obst_{i-1}$.

6.3 Multiple obstacles

As we have presented in the previous chapter, when considering a multiple obstacles scenario, we assume that the various obstacles are placed in such a way that allows for separate analysis - in terms of our avoidance strategy - of each one. In this case, and because we are trying to find what the vehicle observes at each moment, we can no longer treat each obstacle individually, since there may be situations in where one or more obstacles are occluded by some other obstacle. This situation is illustrated in figure 6.7, where all the obstacles are partially occluded by the obstacle O_1.

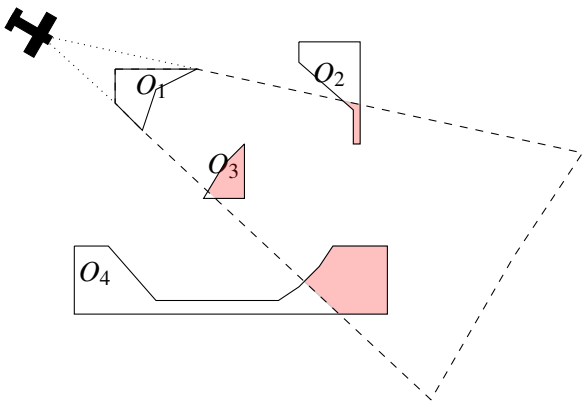


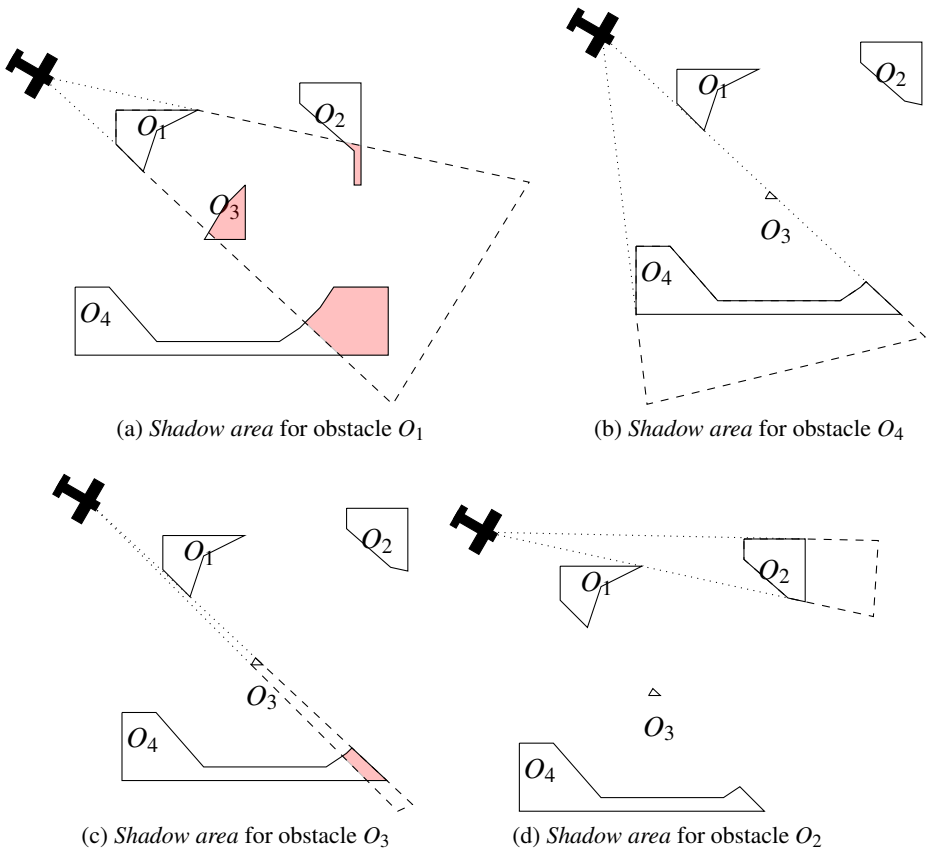
Figure 6.7: Shadow cast by obstacle O_1

The sequence by which the obstacles are analyzed in terms of visibility now becomes important. If, for example, we start by analyzing an arbitrary obstacle, like O_3 instead of O_1, we are implicitly assuming that O_3 is visible even though it lies in the shadow of O_1. One possible solution to avoid these situations is to determine a correct order for the application of the *shadow area* method in terms of decreasing proximity between the vehicle and the obstacles. In this order, we can update the rest of the obstacles, analyzing if these are visible or not given the actual *shadow area* computation.

In order to justify our reasoning, recall the situation presented in figure 6.7. The closest obstacle to the vehicle is O_1, followed by O_4, O_3 and for last, O_2. Since an obstacle that is occluded by some other obstacle should not be considered visible, special attention should be paid to the choice of the value of the *extrusion* distance. In order to prevent these situations, choosing this distance to be greater than or equal to the maximum distance between any two vertices of the polygons in the scenario, we try to ensure that an obstacle that is occluded by some other will be (even if partially) in the latter's *shadow polygon*. Thus, only after determining the *shadow area* of the closest obstacle (represented by the dashed line in figure 6.7), we should compute the shadow of the remaining obstacles. However, it should be noticed that we can not guarantee that this procedure will result in all invisible parts not been considered. One example of such a case is

shown in figure 6.8(b), where a part of O_4 was deemed visible, but in reality isn't as it is occluded by O_3 (6.8(c)).

As we can see in figure 6.7, the shaded areas are not visible after the *shadow area* computation for the nearest obstacle, O_1 , something which will have implications on the *shadow area* computation for the remaining polygons. To take the current *shadow area* computation into account when computing the remaining *shadow polygons*, we use the following reasoning - if the intersection between the computed *shadow area* and a given obstacle is equal to the obstacle itself, then that obstacle is totally inside the *shadow area* and thus it is not visible to the vehicle so there is no need to compute its *shadow polygon*. If, however, there is no resulting intersection set (the intersection results in the empty set), then the obstacle is not occluded by the actual *shadow area* and so it may be completely visible to the vehicle (unless another *shadow area* occludes it). Finally, if there is a non-empty intersection that at the same time is also not equal to the polygon itself, the obstacle is partially occluded. Being partially occluded, we must compute the shadow cast by its visible part (the complement of the result of the intersection). This process continues until the computation of the *shadow area* of the farthest obstacle takes place (see figure 6.8, where the *extrusion* distance has been decreased for presentation reasons).



(a) *Shadow area* for obstacle O_1

(b) *Shadow area* for obstacle O_4

(c) *Shadow area* for obstacle O_3

(d) *Shadow area* for obstacle O_2

Figure 6.8: Different *shadow areas* for the scenario in figure 6.7

After this process, it is necessary to join all the computed *shadow area*, forming the correspondent avoidance obstacle(s). For the previous example, the avoidance obstacle is presented in figure 6.9. Note that the *extrusion* distance has been decreased for presentation reasons.

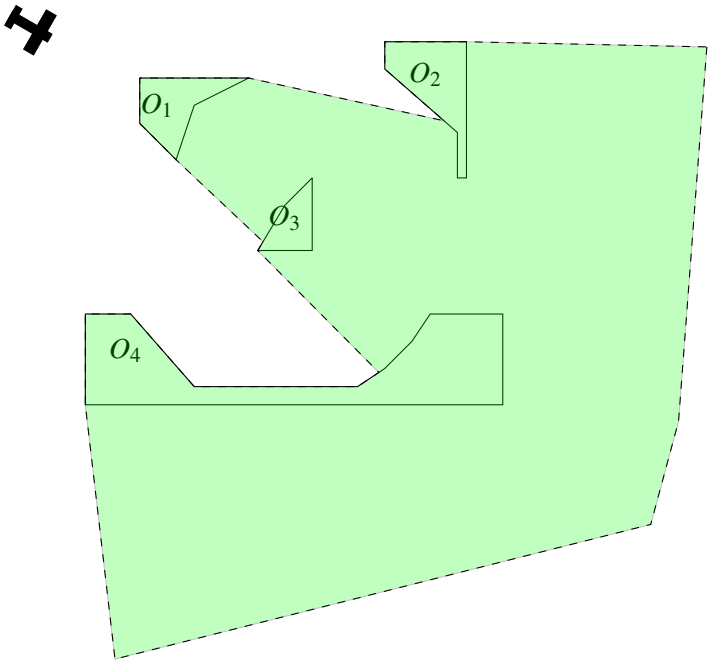


Figure 6.9: Resulting *shadow area*

6.4 Reduced visibility

So far, we have assumed that the vehicle has omnidirectional sensing, with a range greater than or equal to twice the vehicle's minimum turning radius, r_c. This assumption was made in order to guarantee that the vehicle does not enter places from which it can not exit without colliding. One of these situations is shown in figure 6.10.

In figure 6.10, the vehicle does not have complete knowledge about the obstacle and, because of its reduced visibility (represented by the shaded blue area in front of the vehicle), it is not capable of analyzing correctly the viability of both avoidance maneuvers. In the first instant when the avoidance controller detects that it is necessary to start an avoidance maneuver (see figure 6.10(c)), the vehicle is in a unsafe state, since both maneuvers will result in a collision. This way, a "wide-aperture" sensor is fundamental to our approach.
We finish with a remark, putting together the most important parts in this work.

Remark 2 *Consider the vehicle described by definition 3.22 and controlled by the hybrid automaton shown in figure 5.4. Also, assume a partially known world in which theorem 3 holds. If*

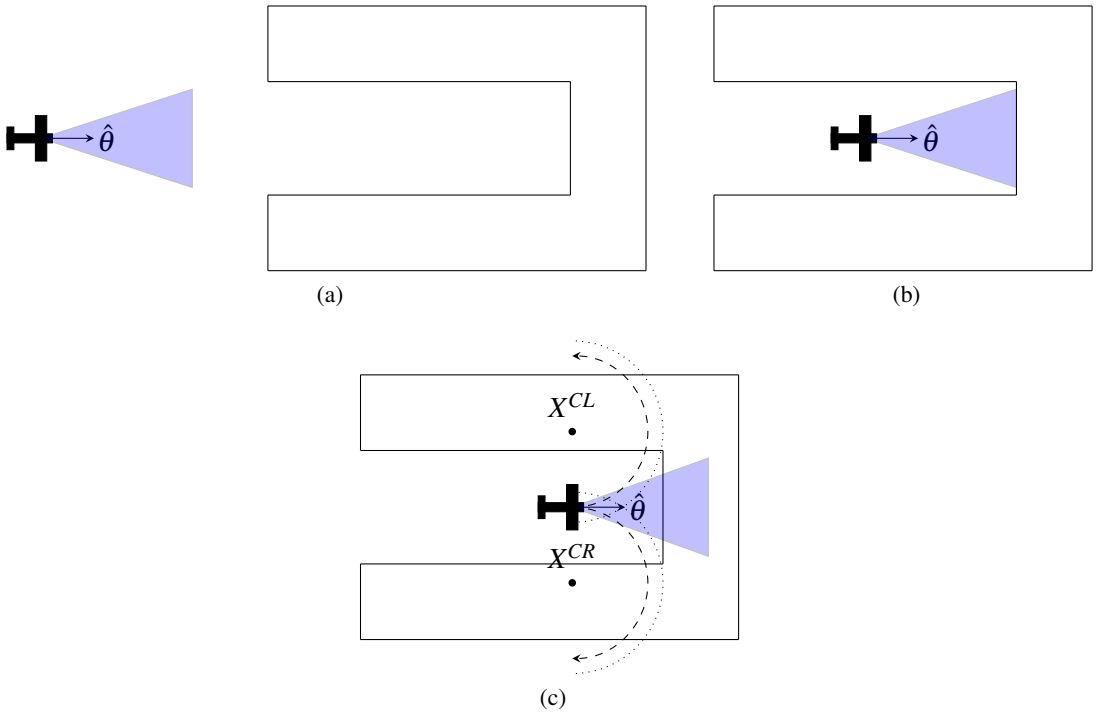


Figure 6.10: Reduced visibility problem

the vehicle has omnidirectional sensing, with a range greater than or equal to twice the vehicle's minimum turning radius, r_c, we can guarantee that the vehicle does not enter unsafe positions.

Chapter 7

Simulation results

Considering the unicycle model described by definition 3.22 and different obstacles as the ones described in previous chapters, we present some simulations showing our results. We will begin with polygonal obstacles, studying the case where we have a simple wall as an obstacle, followed by convex and non-convex obstacles. After this we will study spline-based obstacles in the same sequence, first the simple curve, followed by convex and non-convex sets. For last, we will show a multiple obstacle scenario. All these situations are shown first assuming a completely known environment and then assuming a partially known world. The latter case will only be treated for polygonal obstacles. All the avoidance maneuvers are represented by a dashed (red) line, while the normal (where the mission controller is driving the vehicle as opposed to the avoidance controller) trajectories are drawn in a full (blue) line.
In the simulations, the vehicle travels from an initial state, $x_0 = [x_1, x_2, \theta]^T$, to a target point, $x_T = [x_{1T}, x_{2T}]^T$. Recall that we let the vehicle travel with the same constant linear velocity ($U = 300m/s$), and a maximum turning speed of $W_{max} = 0.3rad/s$. Other relevant parameters are the time step of $\tau = 10ms$, the delay modeling the roll dynamics ($t_d = 2\tau$) and the tolerance parameter $\varepsilon = (t_d + 7\tau)U$.

Example 1 (A completely known world - Simple Wall) *This first example shows the vehicle departing from the initial state $x_0 = [0, 2000, \frac{\pi}{4}]^T$ (represented by a star), and going towards a simple wall (see figure 7.1). The target, located at $x_T = [2000, -1500]^T$ is represented by the airplane figure.*

Example 2 (A completely known world - Polygonal obstacles) *After showing that our approach works for a simple wall, we move on to a different type of obstacles. We present two different examples with convex polygons (figures 7.2 and 7.3) and then two more examples with non-convex polygons (figures 7.4 and 7.5).*

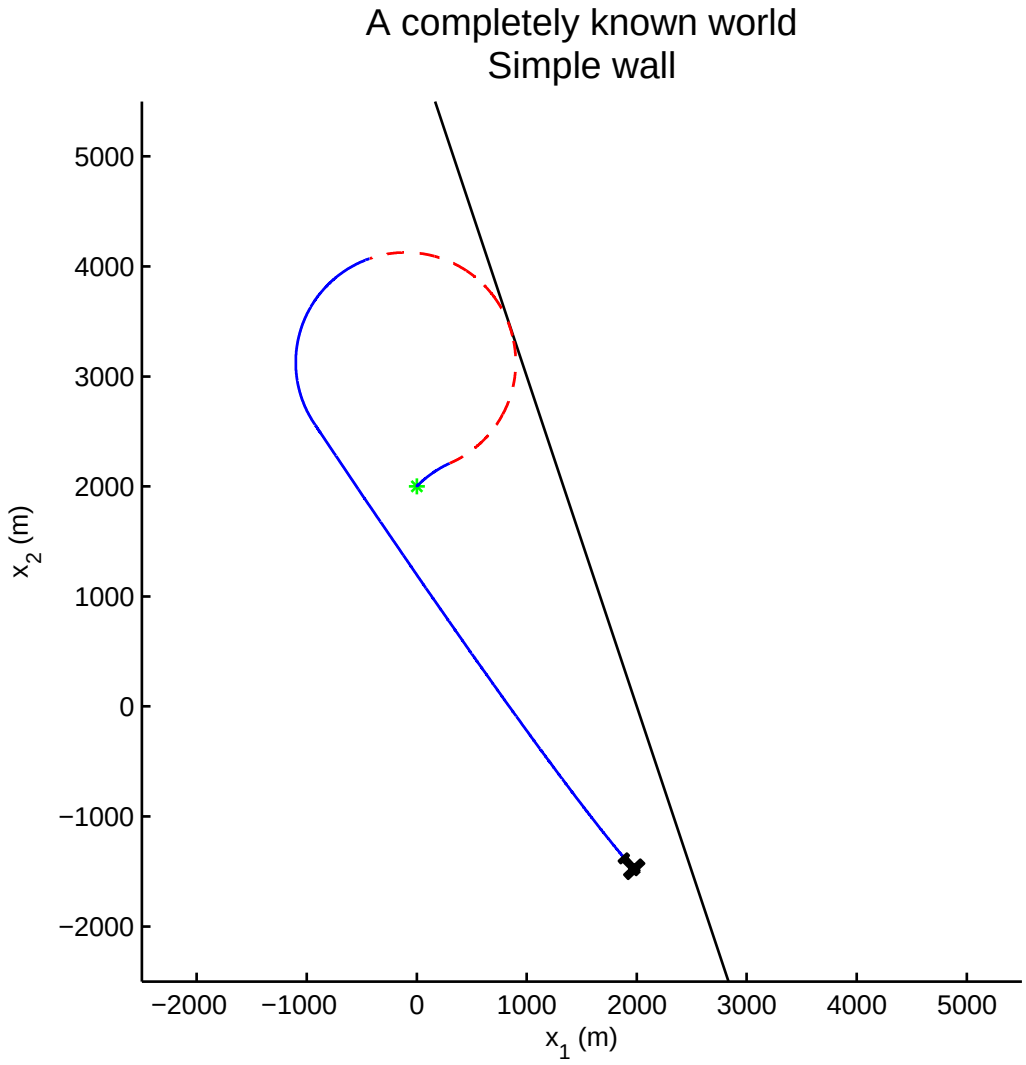


Figure 7.1: Example 1 - note that $x_0 = [0, 2000, \frac{\pi}{4}]^T$ and $x_T = [2000, -1500]^T$

Example 3 (A completely known world - Spline-based obstacles) *Moving into another type of obstacles, we start by presenting a simple curve as our obstacle, defined by a single spline (figure 7.6).*

Example 4 (A completely known world - Convex and non-convex spline-based sets) *Consider now convex and non-convex sets defined by splines as our obstacles. This sequence is shown in figures 7.7 and 7.8 for the convex obstacles and figures 7.9 and 7.10 for the non-convex ones.*

Example 5 (A completely known world - Multiple obstacles scenario) *Consider now a multiple obstacles scenario. In figure 7.11 we show a scenario only with polygonal obstacles, while in*

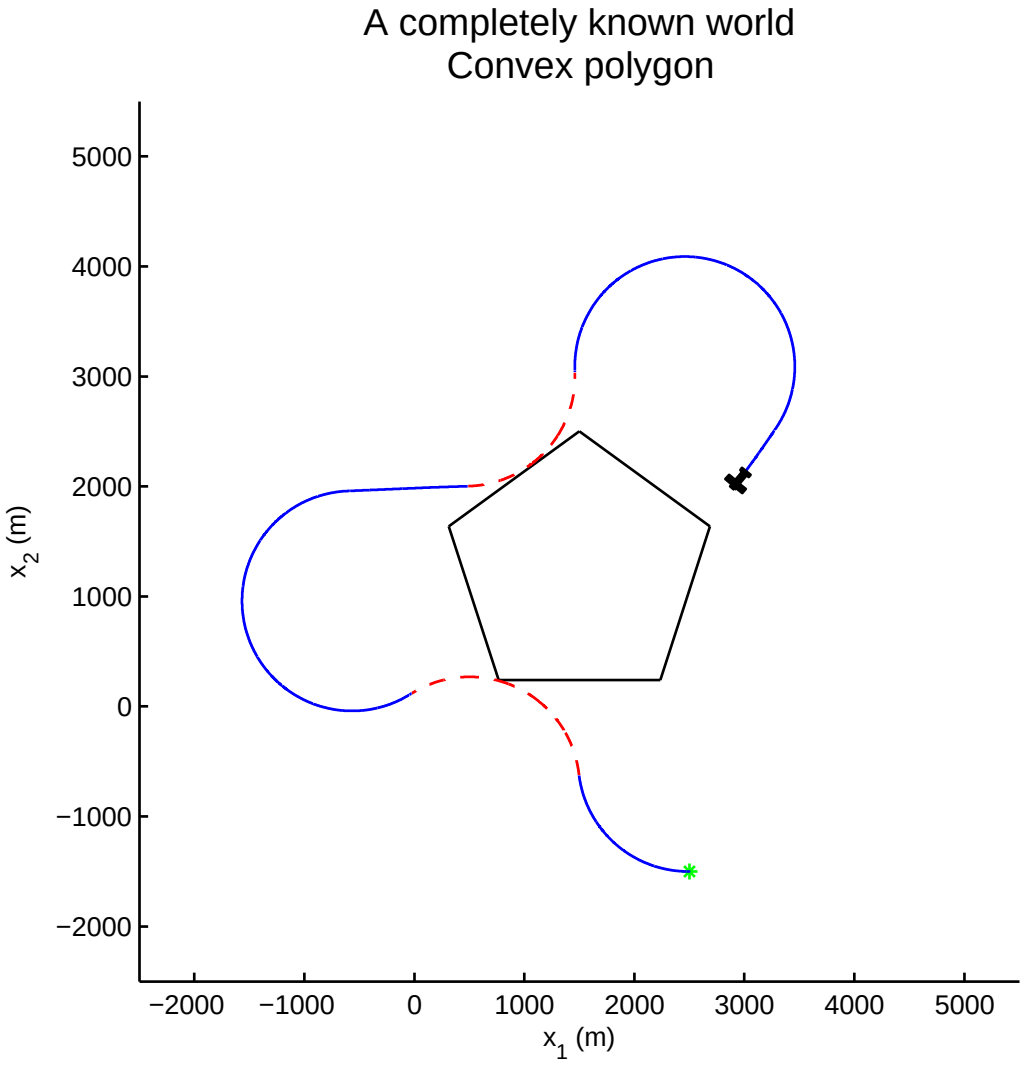


Figure 7.2: Example 2.1 - note that $x_0 = [2500, -1500, \pi]^T$ and $x_T = [2900, 2000]^T$

figure 7.12 we use spline-based obstacles. We then present in figures 7.13 and 7.14 a combination of polygonal and spline-based obstacles.

Example 6 (A partially known world - Polygonal obstacles) *Now that we have shown our results for the case where the world is completely known, let us repeat some of the above examples for the case where this knowledge is acquired as the vehicle travels. We let the initial and final positions be the same as before, representing the final known obstacle(s) by the cyan dash-dotted line. In figures 7.15, 7.16 and 7.17 the vehicle is able to acquire a complete knowledge of the real obstacles, whereas in figure 7.18 the vehicle's final knowledge (still) represents an over-approximation of the real obstacle.*

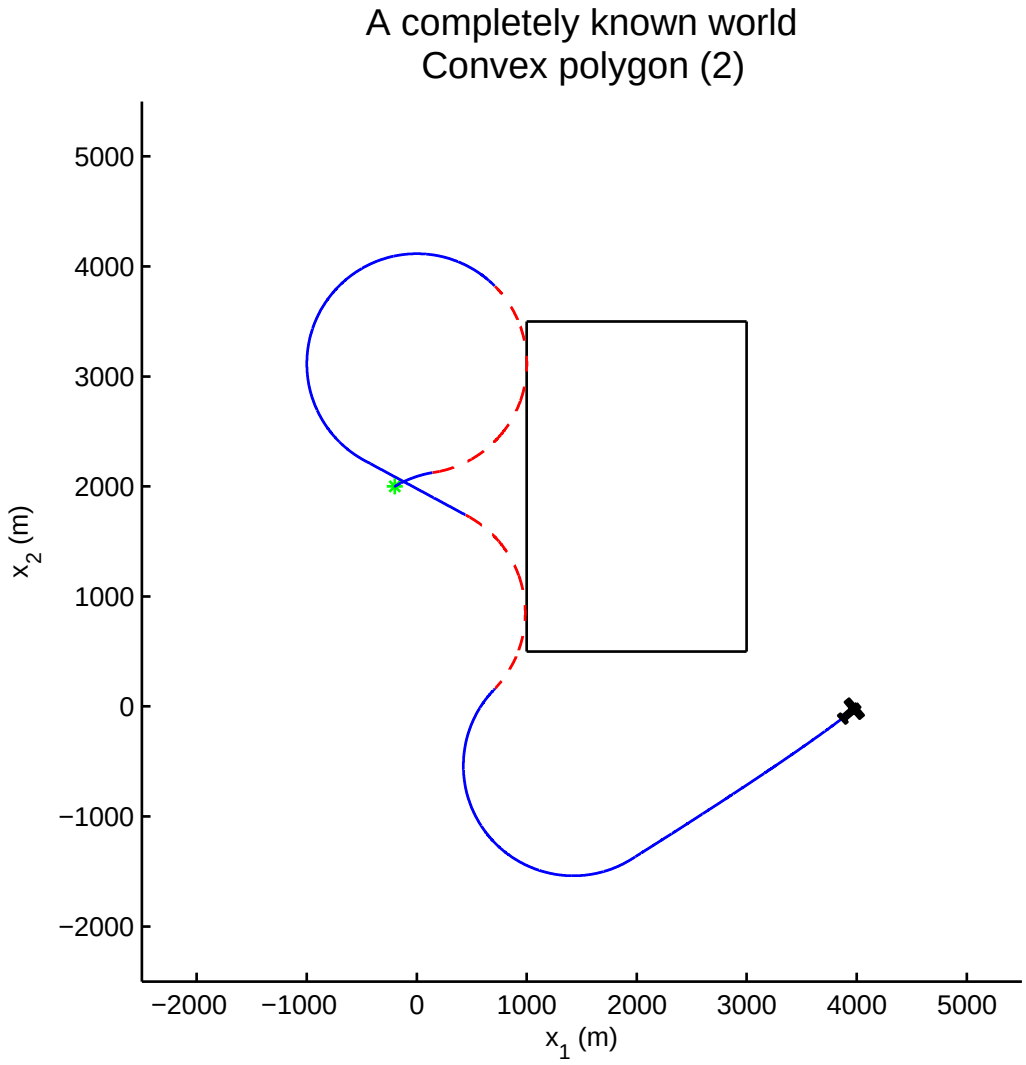


Figure 7.3: Example 2.2 - note that $x_0 = [-200, 2000, \frac{\pi}{6}]^T$ and $x_T = [4000, 0]^T$

Example 7 (A partially known world - Multiple obstacles scenario) *For last, we show the case where the vehicle moves in an environment with polygonal obstacles (figures 7.19 and 7.20).*

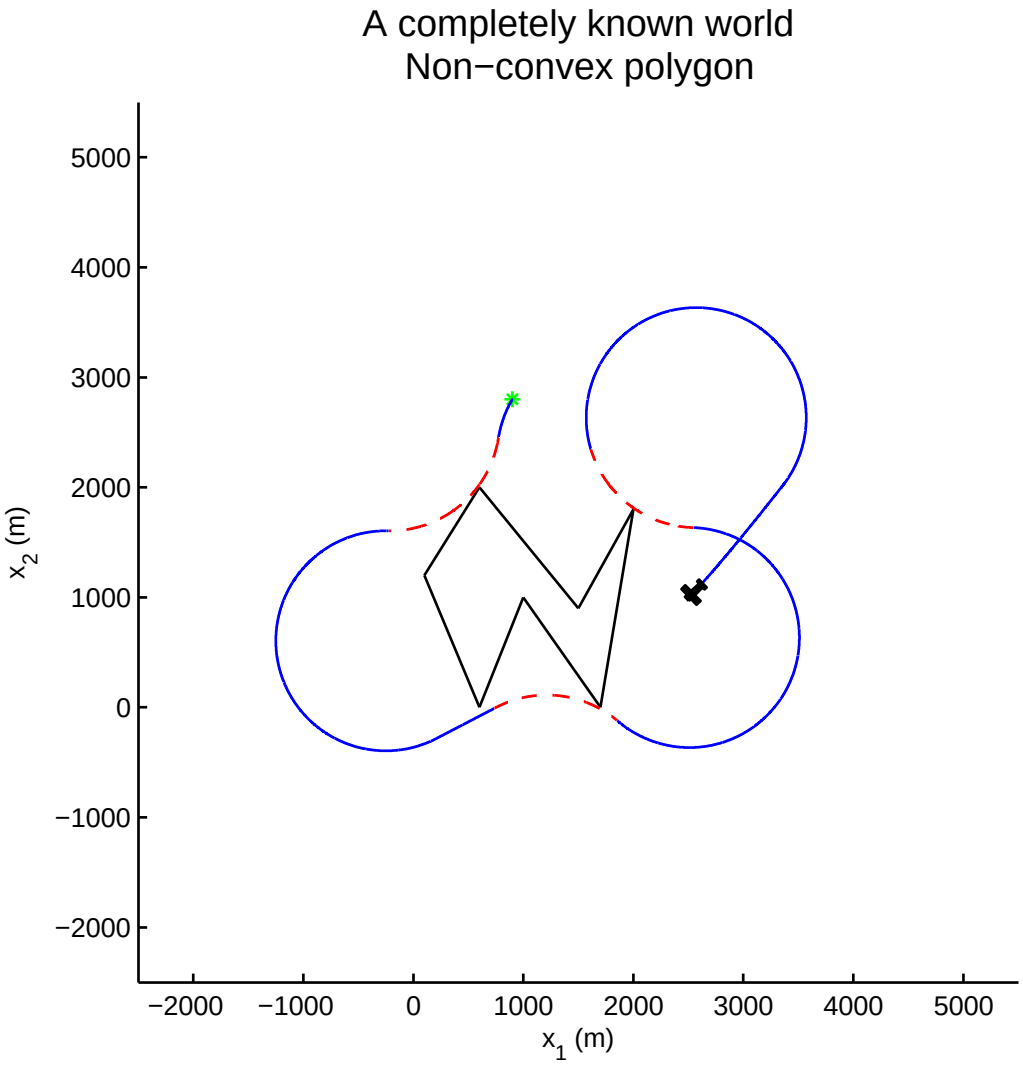


Figure 7.4: Example 2.3 - note that $x_0 = [900, 2800, \frac{4\pi}{3}]^T$ and $x_T = [2500, 1000]^T$

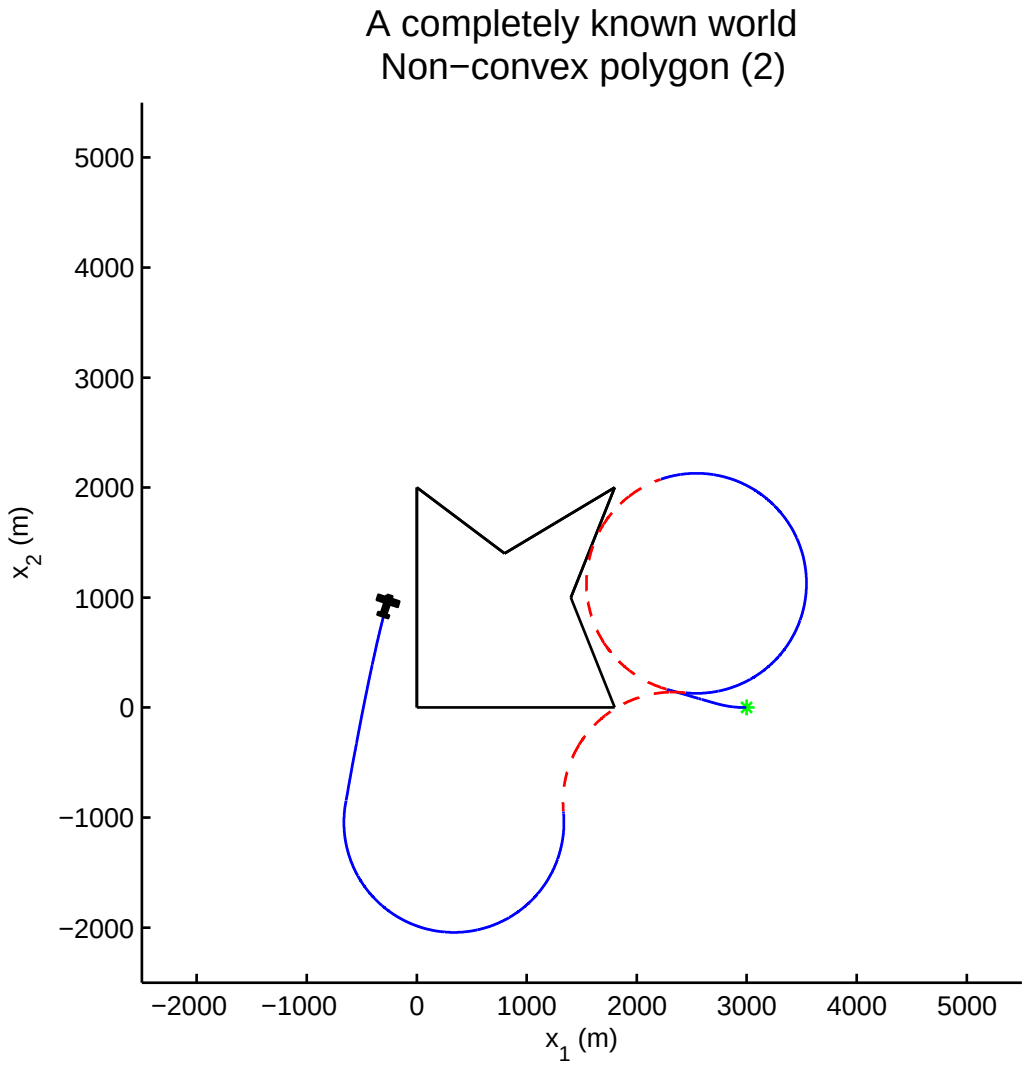


Figure 7.5: Example 2.4 - note that $x_0 = [3000, 0, \pi]^T$ and $x_T = [-250, 1000]^T$

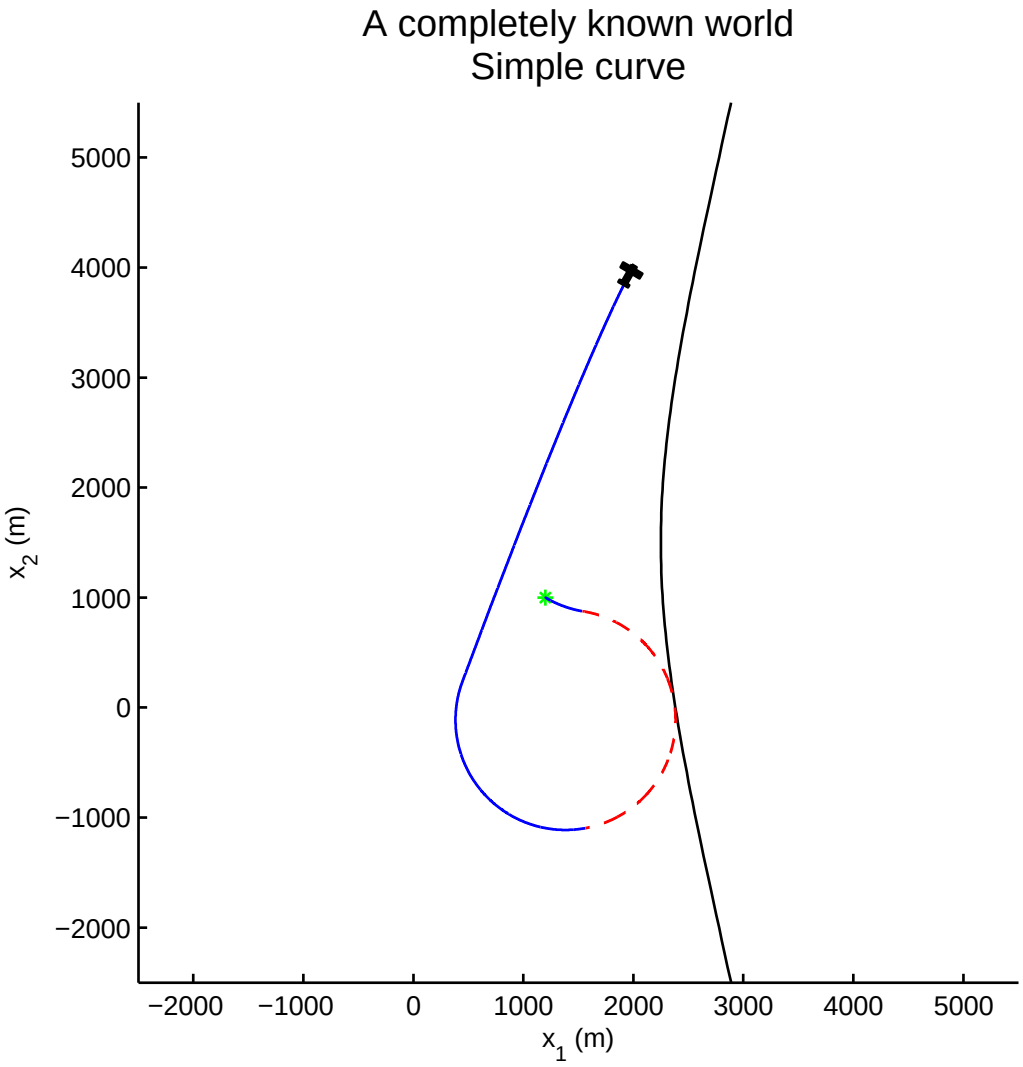


Figure 7.6: Example 3 - note that $x_0 = [1200, 1000, \frac{-\pi}{6}]^T$ and $x_T = [2000, 4000]^T$

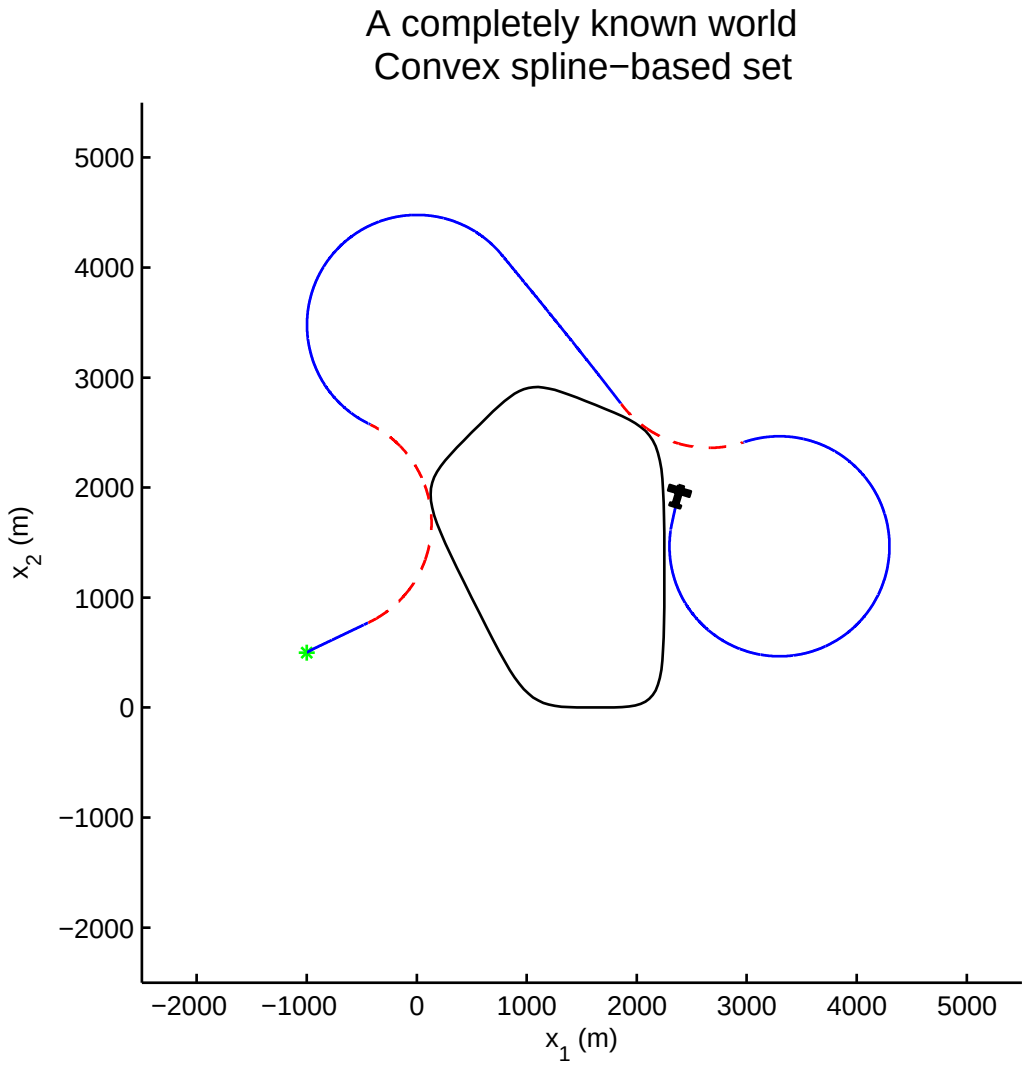


Figure 7.7: Example 4.1 - note that $x_0 = [-1000, 500, \frac{\pi}{6}]^T$ and $x_T = [2400, 2000]^T$

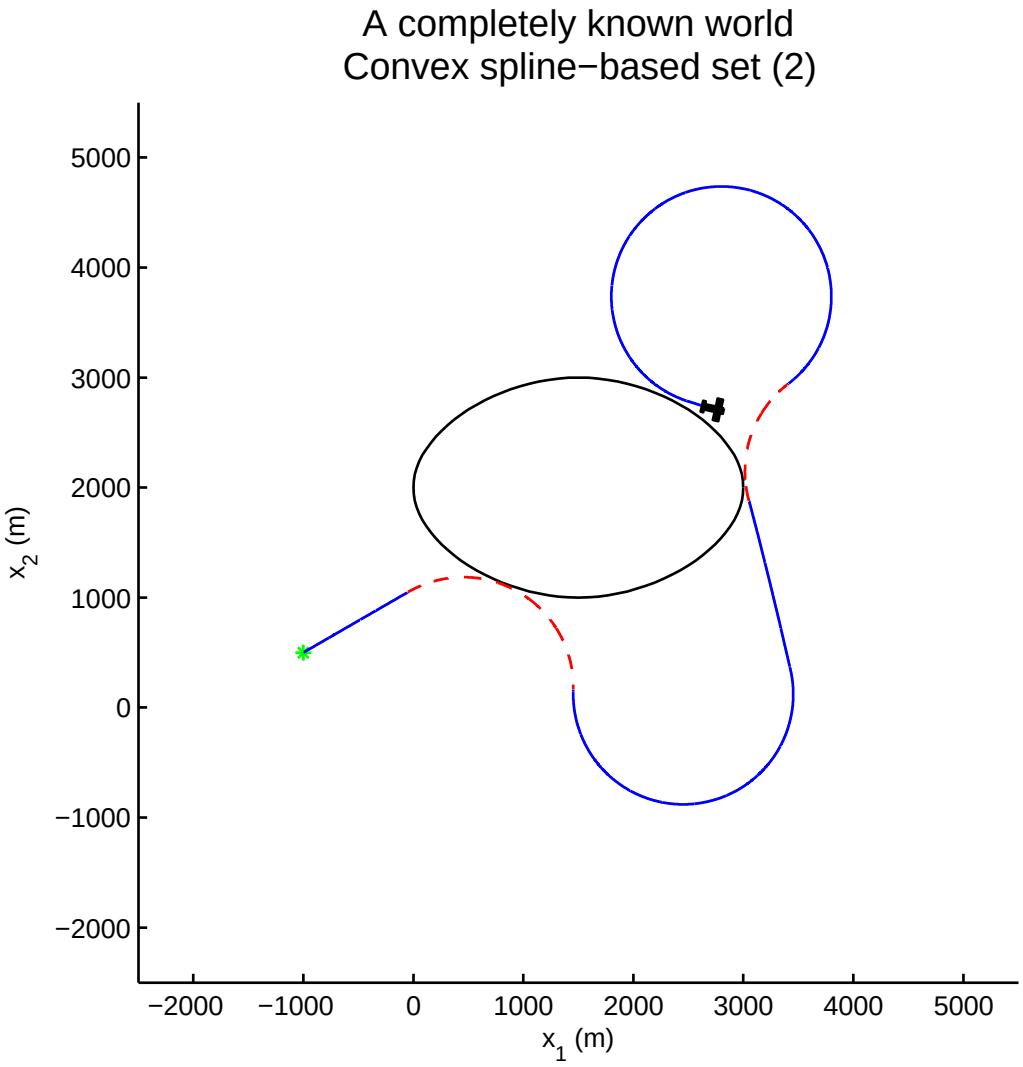


Figure 7.8: Example 4.2 - note that $x_0 = [-1000, 500, \frac{\pi}{6}]^T$ and $x_T = [2800, 2700]^T$

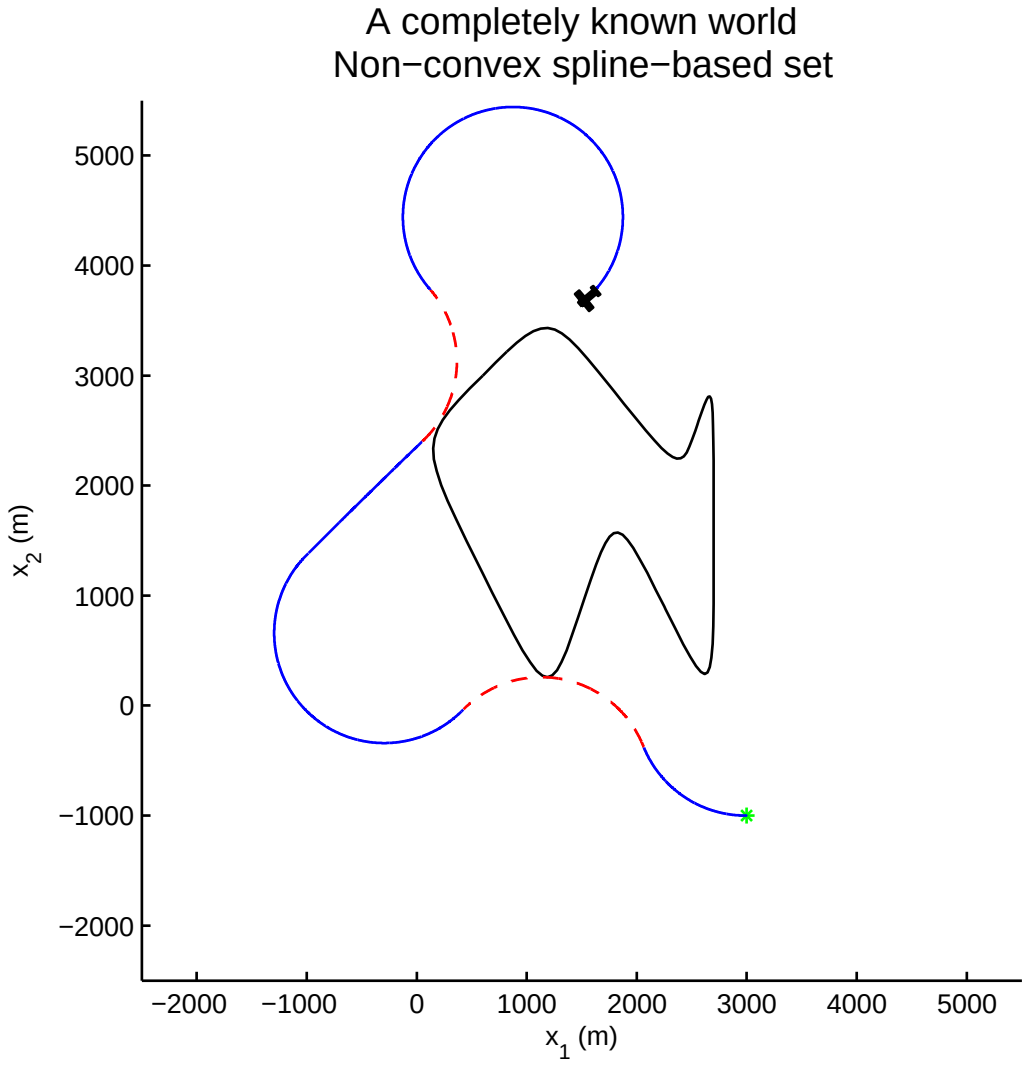


Figure 7.9: Example 4.3 - note that $x_0 = [3000, -1000, \pi]^T$ and $x_T = [1500, 3700]^T$

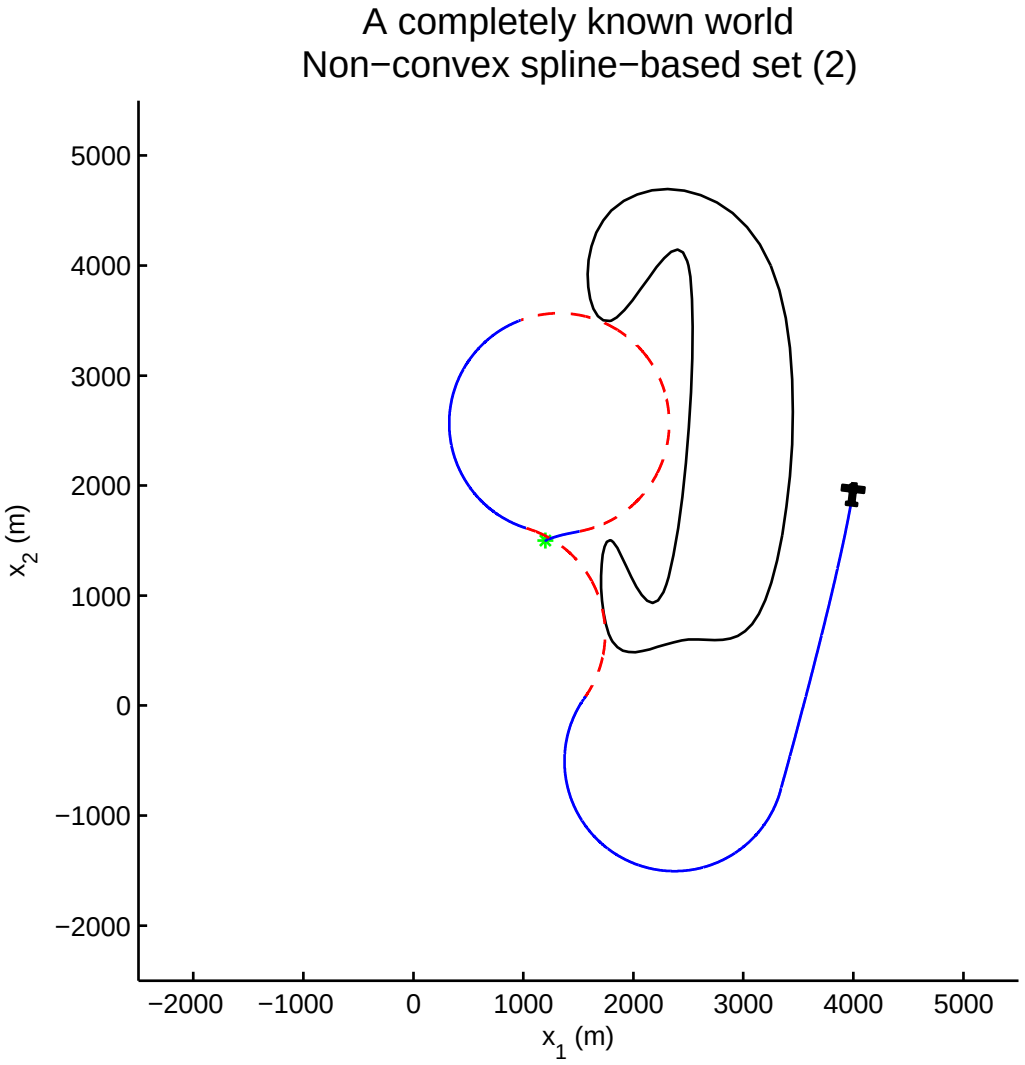


Figure 7.10: Example 4.4 - note that $x_0 = [1200, 1500, \frac{\pi}{8}]^T$ and $x_T = [4000, 2000]^T$

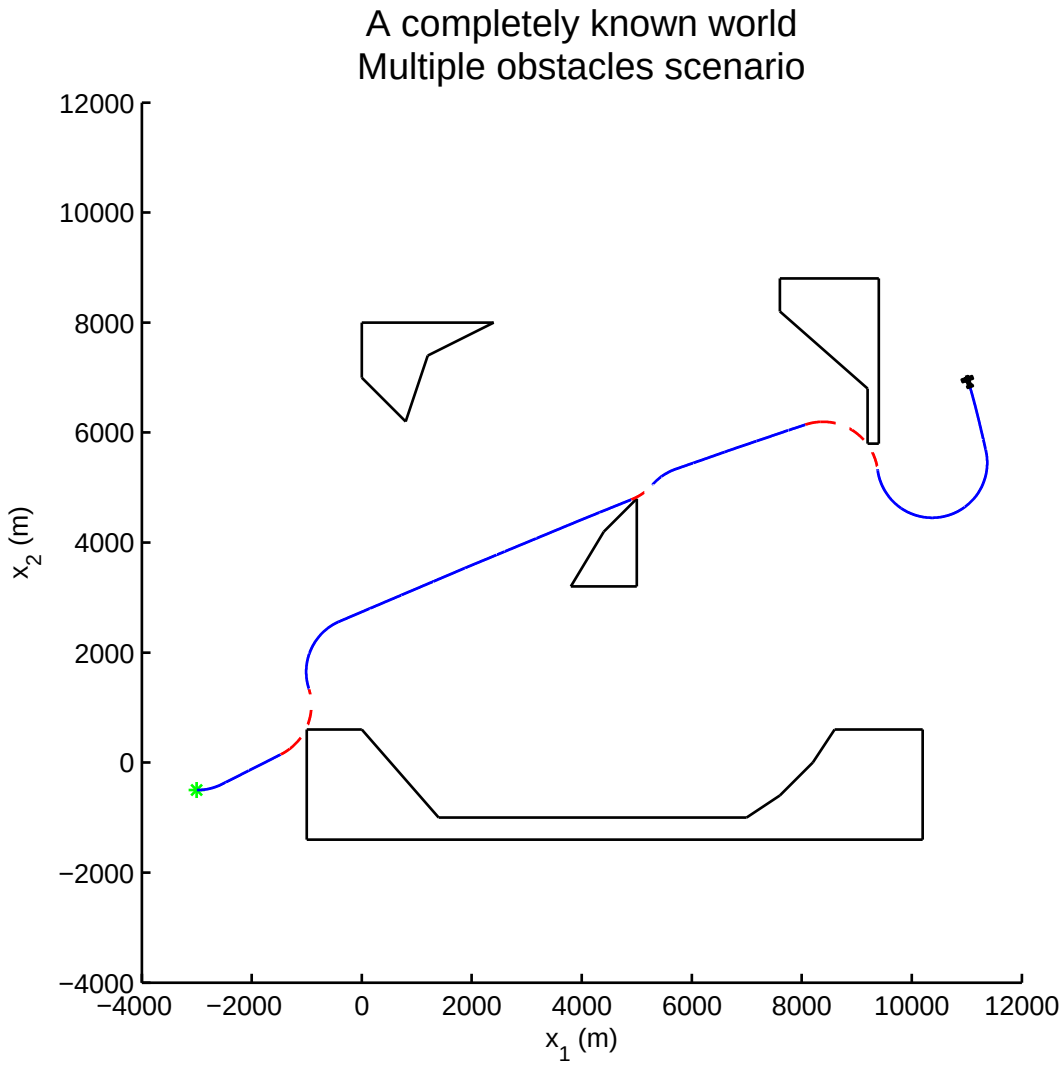


Figure 7.11: Example 5.1 - note that $x_0 = [-3000, -500, 0]^T$ and $x_T = [11000, 7000]^T$

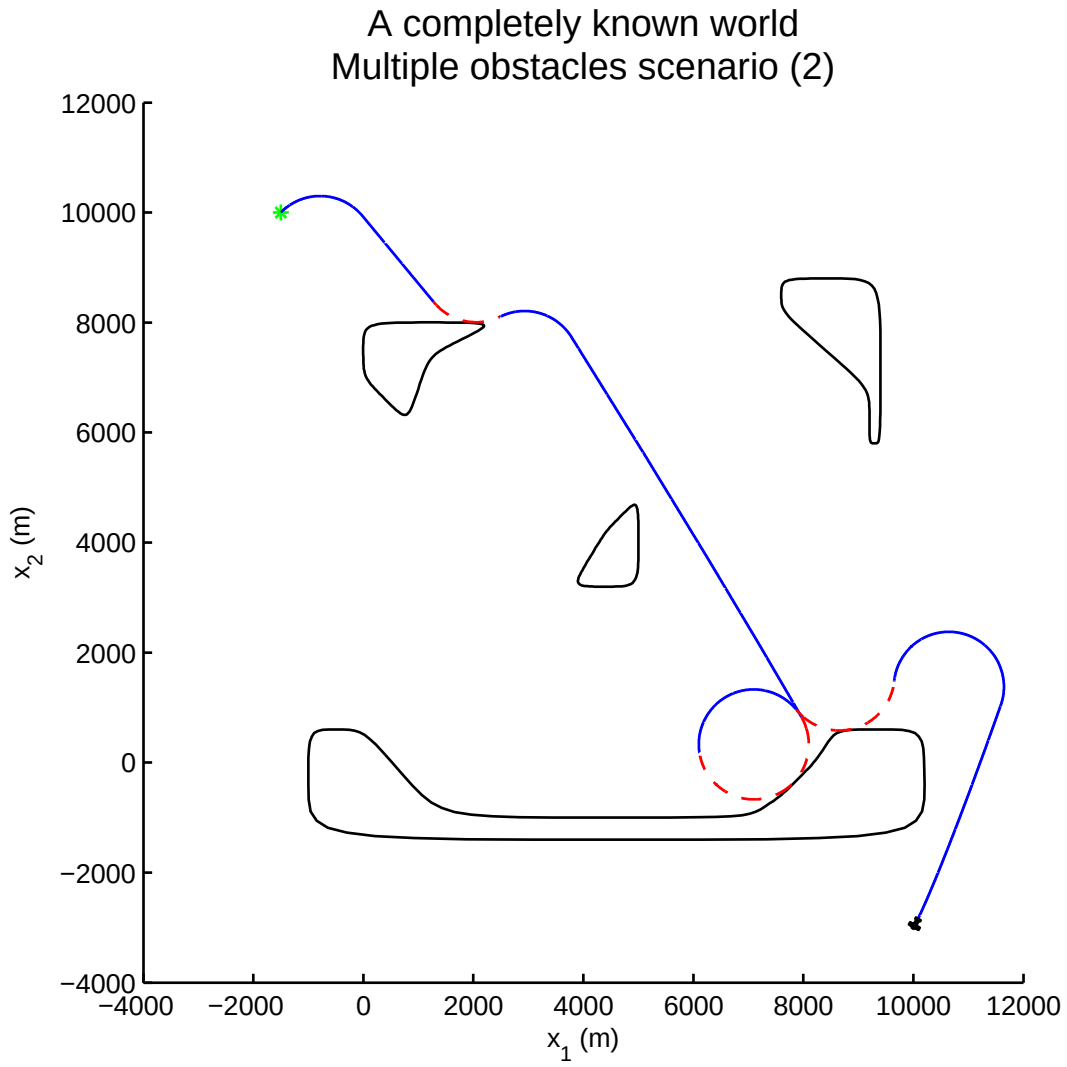


Figure 7.12: Example 5.2 - note that $x_0 = [-1500, 10000, \frac{\pi}{4}]^T$ and $x_T = [10000, -3000]^T$

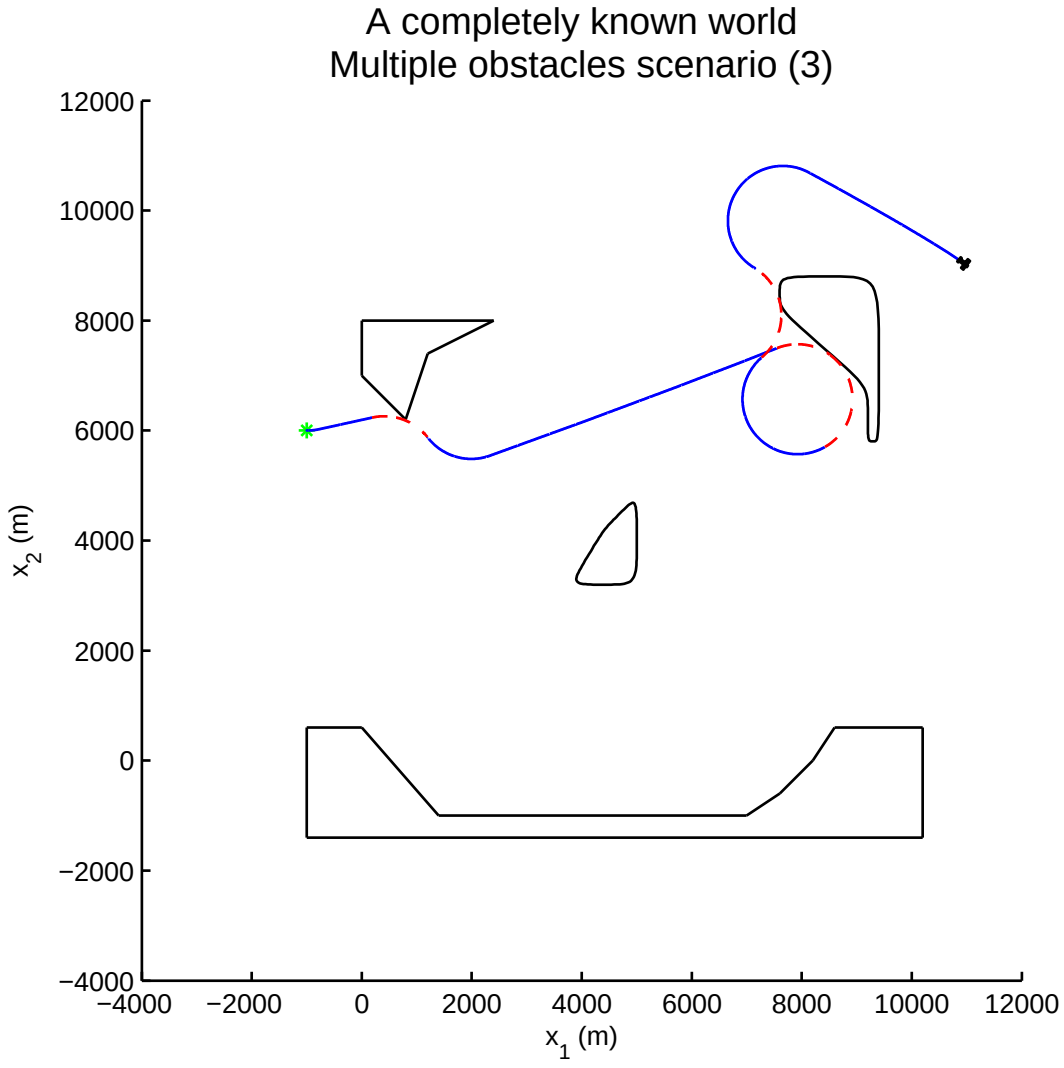


Figure 7.13: Example 5.3 - note that $x_0 = [-1000, 6000, 0]^T$ and $x_T = [11000, 9000]^T$

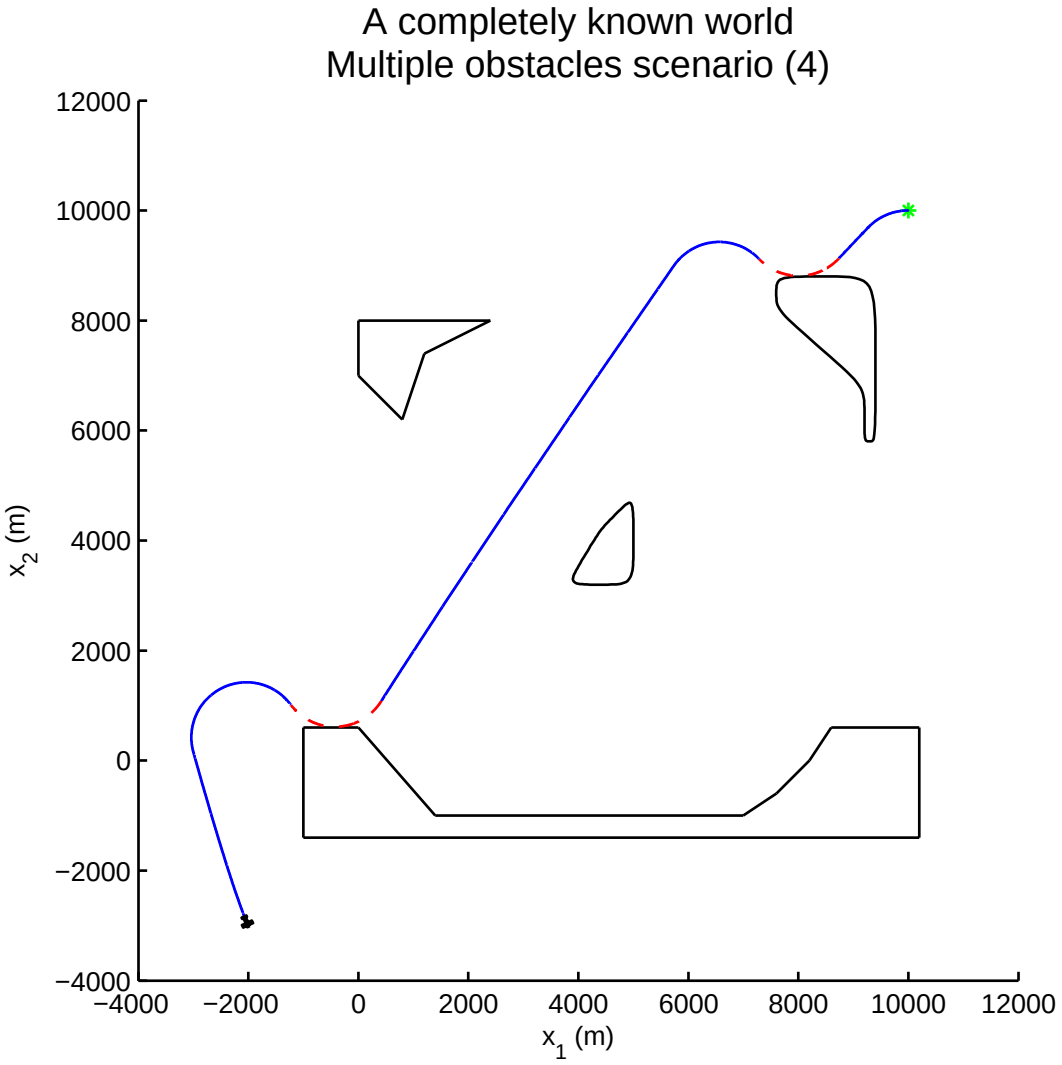


Figure 7.14: Example 5.4 - note that $x_0 = [10000, 10000, \pi]^T$ and $x_T = [-2000, -3000]^T$

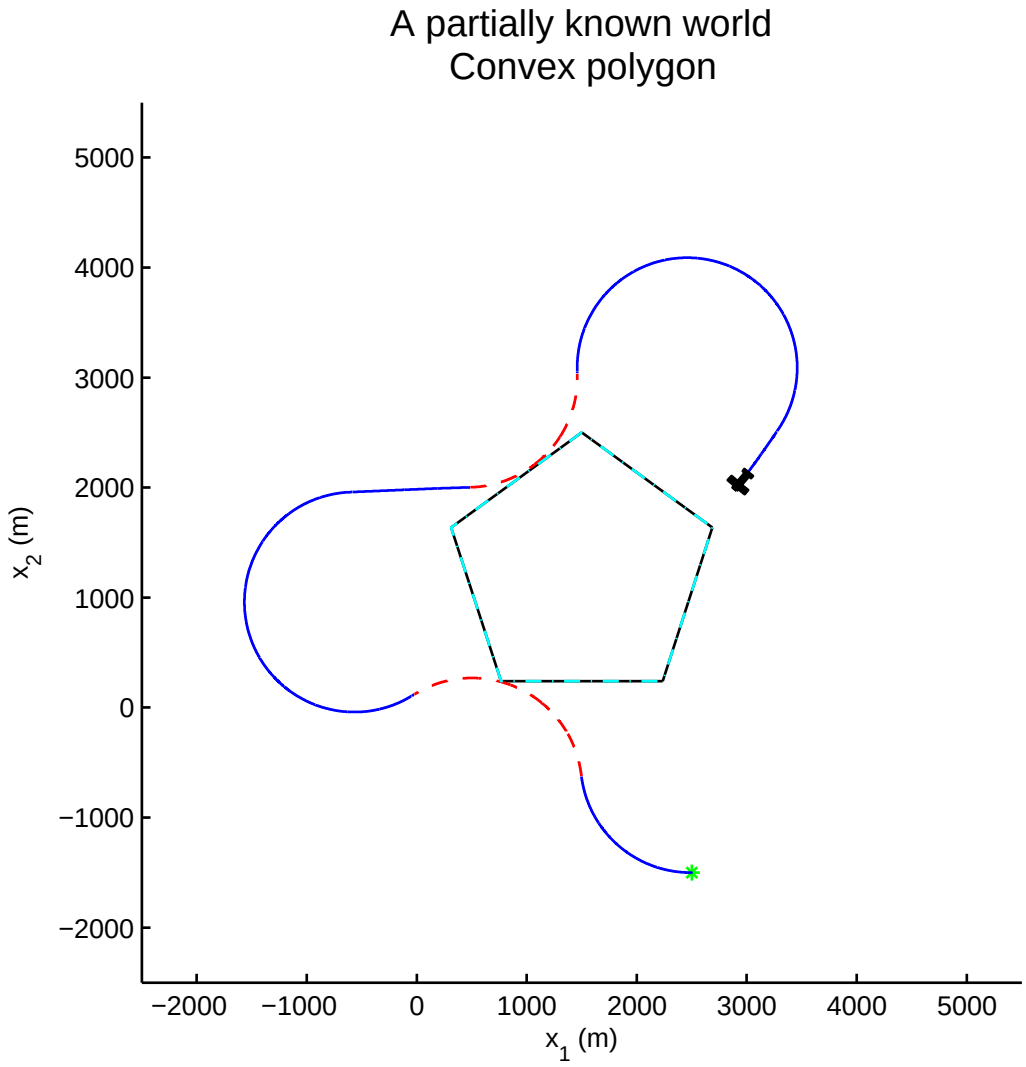


Figure 7.15: Example 6.1 - note that $x_0 = [2500, -1500, \pi]^T$ and $x_T = [2900, 2000]^T$

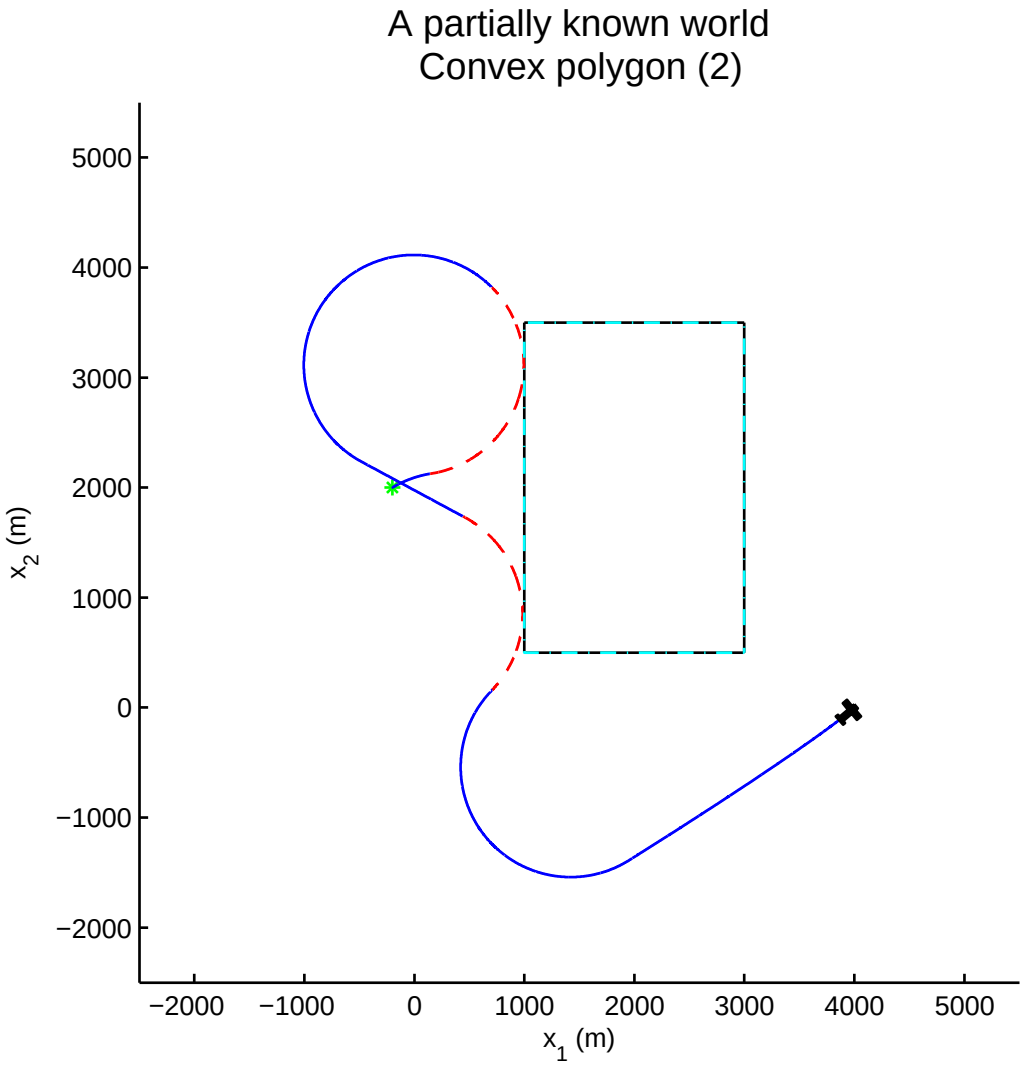


Figure 7.16: Example 6.2 - note that $x_0 = [-200, 2000, \frac{\pi}{6}]^T$ and $x_T = [4000, 0]^T$

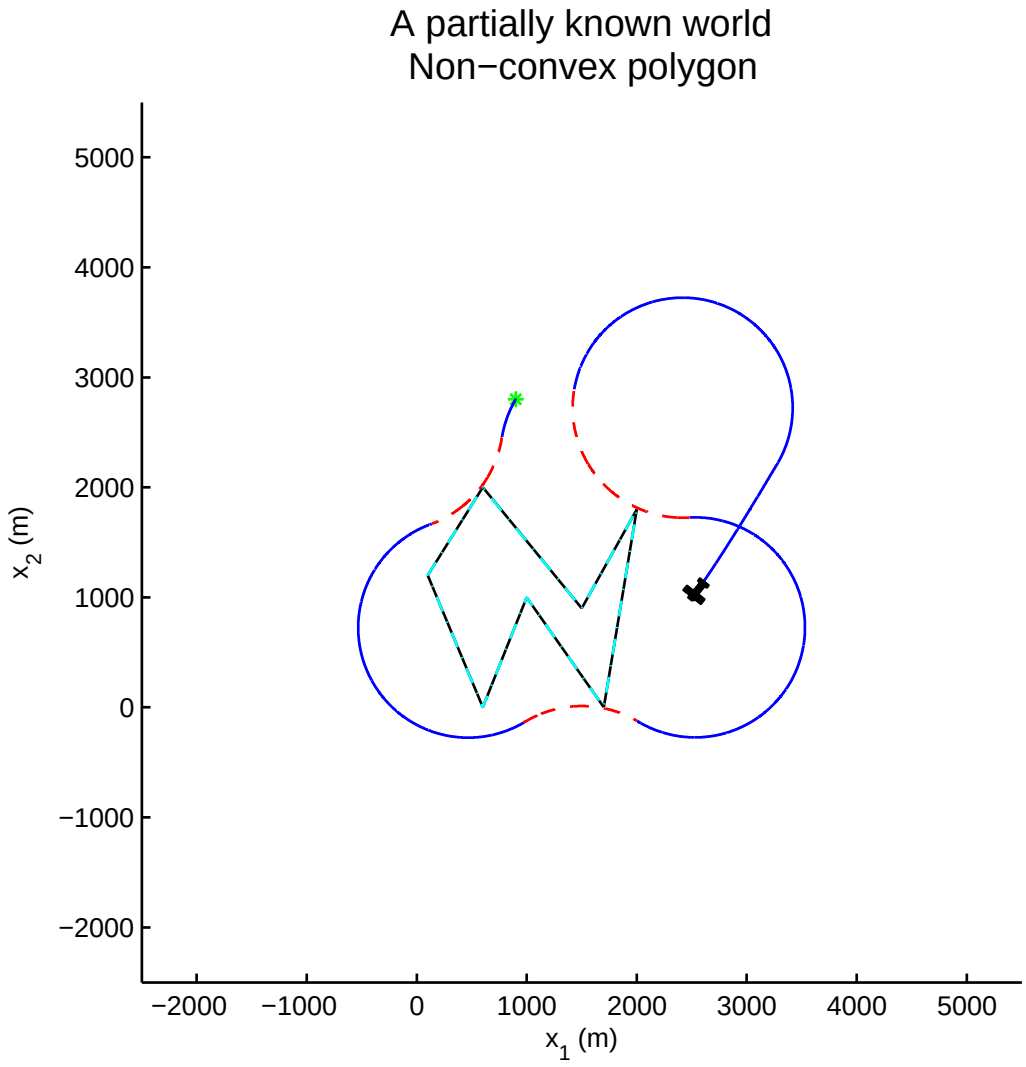


Figure 7.17: Example 6.3 - note that $x_0 = [900, 2800, \frac{4\pi}{3}]^T$ and $x_T = [2500, 1000]^T$

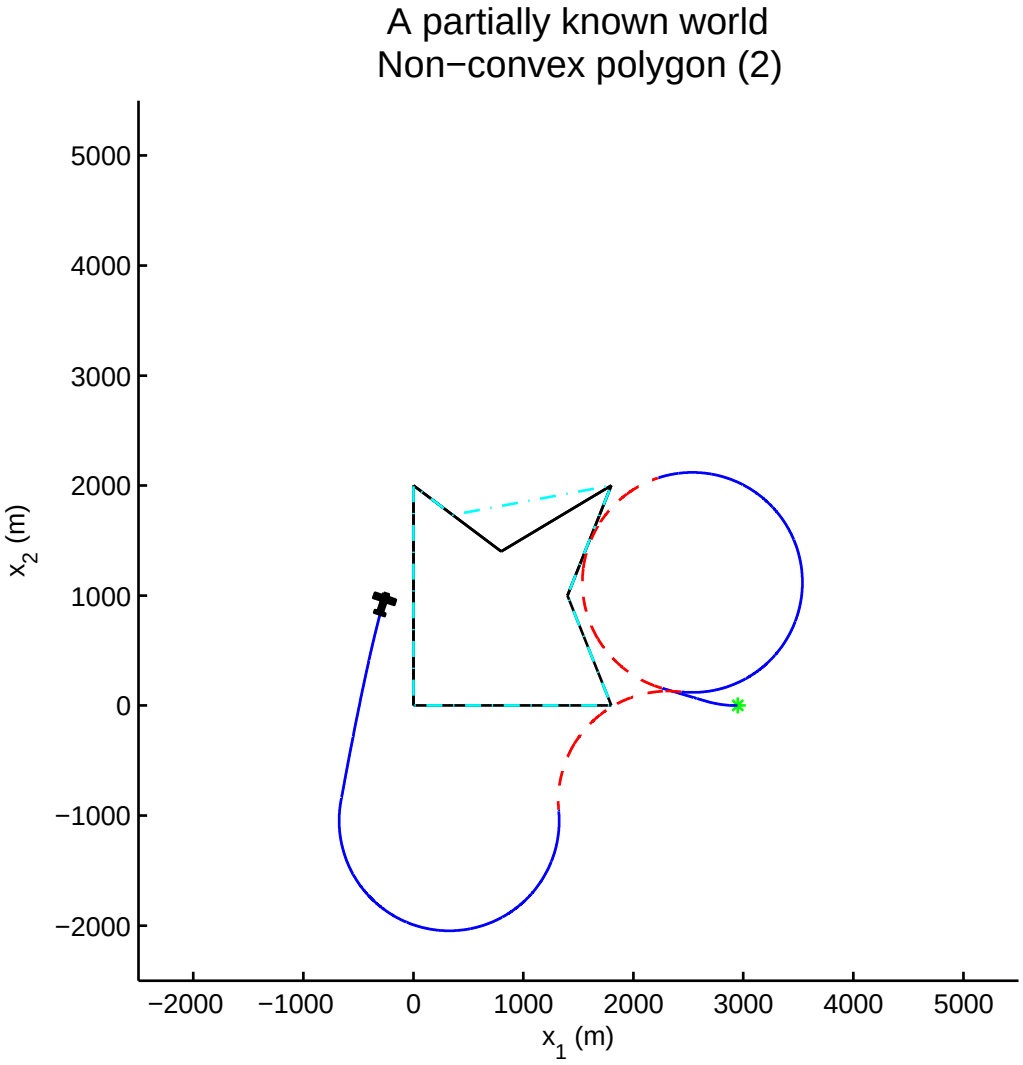


Figure 7.18: Example 6.4 - note that $x_0 = [3000, 0, \pi]^T$ and $x_T = [-250, 1000]^T$

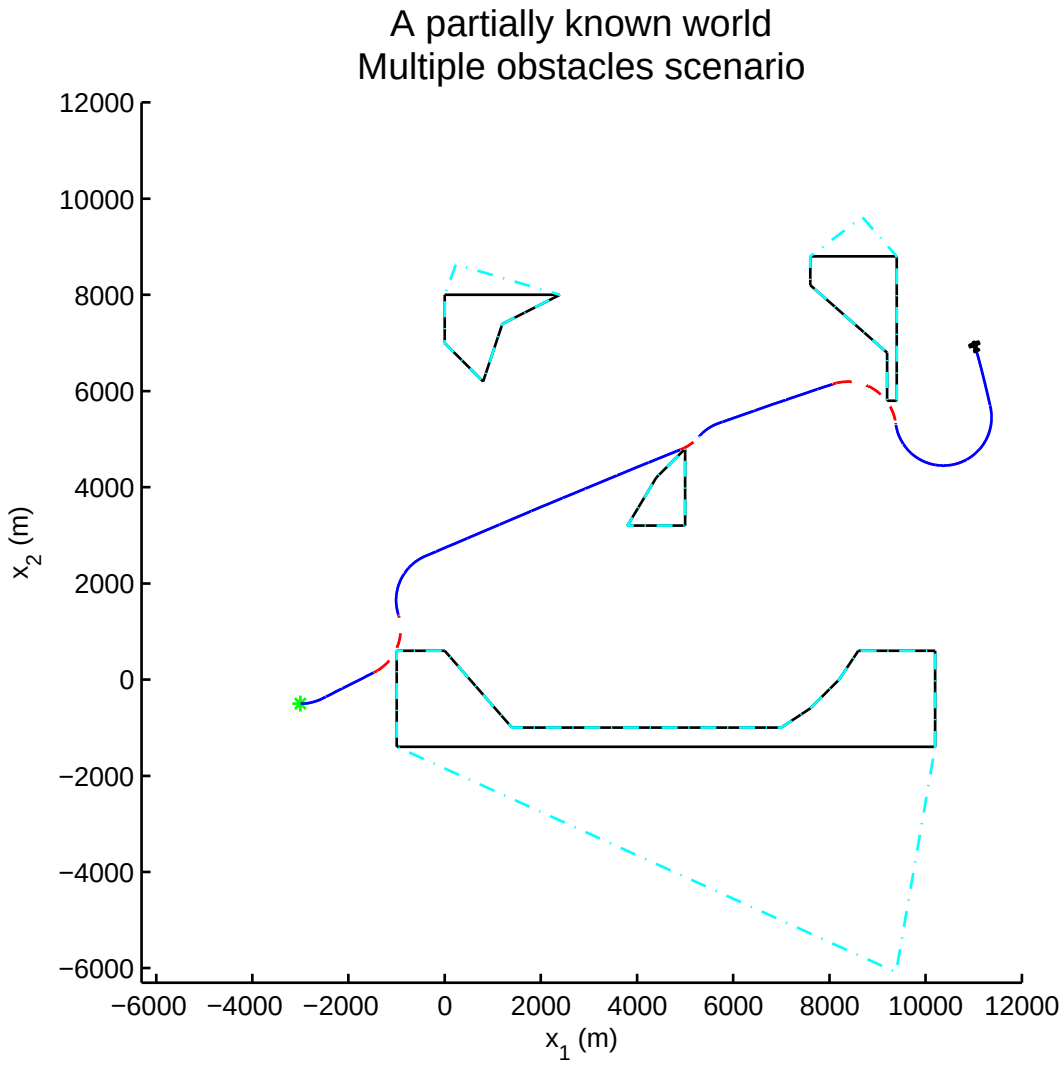


Figure 7.19: Example 7.1 - note that $x_0 = [-3000, -500, 0]^T$ and $x_T = [11000, 7000]^T$

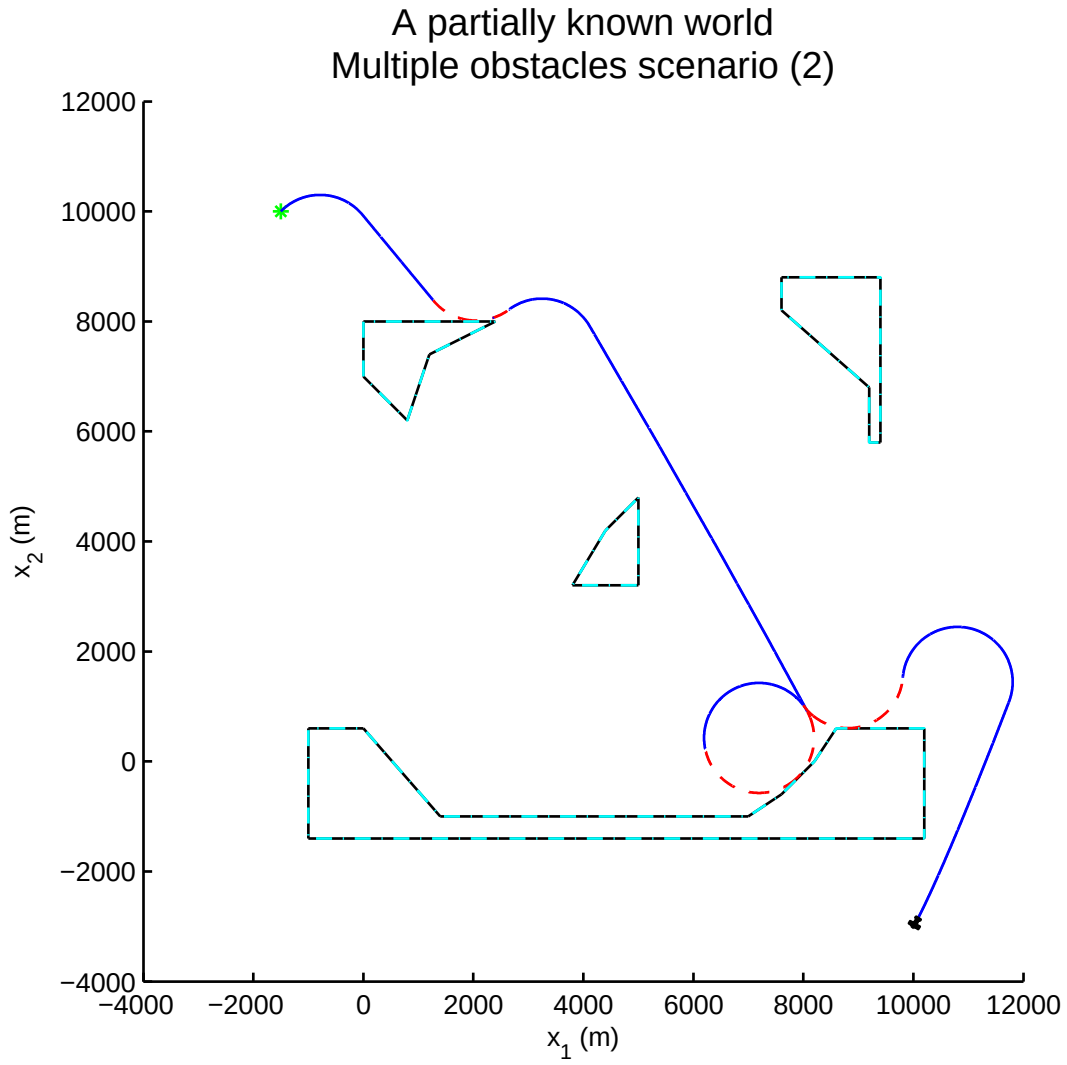


Figure 7.20: Example 7.2 - note that $x_0 = [-1500, 10000, \frac{\pi}{4}]^T$ and $x_T = [10000, -3000]^T$

Chapter 8

Conclusions and future work

The main goal of this thesis was to develop a simple controller capable of dealing with realistic obstacles in a 2D scenario. In order to accomplish this, we studied the avoidance problem for convex and non-convex polygons as well as spline-based sets. Then, we introduced a second topic: analyzing the case where the vehicle has no previous information about the obstacles that it should avoid; instead, its knowledge increases as it travels. Both scenarios were successfully studied, and different solutions were proposed for both problems. It should be mentioned that for the case where there is no previous information about the surrounding environment, the vehicle's sensing capabilities play a vital role in the ability to avoid unsafe states.

The initial safe controller that we implemented only took the vehicle's position into account, without considering its orientation. The weakest point of this controller is that it keeps the vehicle from entering certain safe states, as the avoidance maneuver takes place sooner than it should. For this reason, we modified our initial safe controller so that it takes the vehicle's orientation into account. Although we did not prove that our controller results in the maximal safe set, we can say that, based on our simulation results, our approach has brought important contributions for the avoidance problem, in the sense that it uses hard to implement concepts, like reachable sets and invariance, and it simplifies them with simple geometric assumptions. We should also mention that there are cases where our controller may fail. These particular cases happen when the vehicle is approaching non-differentiable points of the obstacles. Due to discretization, the avoidance maneuver is considered viable, as the computed test points are outside the obstacle, however, between two consecutive time steps, where no test points were computed, the avoidance maneuver intercepts the obstacle and so the vehicle collides. Failure may also occur if Newton's method fails to converge.

In terms of future work, we are interested in studying the partially known world problem to the spline-based sets as this has only been done for the polygonal case. Another important development would be to extend the avoidance problem to the multiple vehicle scenario. Future versions of our approach should also take disturbances into account, which, at the present moment, are not

considered by our approach. For last, a more realistic vehicle model should also be considered, such as a 6 degree-of-freedom aircraft model.

Bibliography

[1] R. Beard and T. McLain, "Multiple uav cooperative search under collision avoidance and limited range communication constraints," in *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 1, pp. 25–30 Vol.1, Dec. 2003.

[2] A. Richards and J. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," in *American Control Conference, 2002. Proceedings of the 2002*, vol. 3, pp. 1936–1941 vol.3, 2002.

[3] T. Schouwenaars, B. DeMoor, E. Feron, and J. How, "Mixed integer programming for multi-vehicle path planning," in *In European Control Conference 2001*, pp. 2603–2608, 2001.

[4] D. Chang, S. Shadden, J. Marsden, and R. Olfati-Saber, "Collision avoidance for multiple agent systems," in *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 1, pp. 539–543 Vol.1, Dec. 2003.

[5] I. Mitchell, *Application of Level Set Methods to Control and Reachability Problems in Continuous and Hybrid Systems*. PhD thesis, Stanford University, August 2002.

[6] C. Tomlin, I. Mitchell, A. Bayen, and M. Oishi, "Computational techniques for the verification of hybrid systems," *Proceedings of the IEEE*, vol. 91, pp. 986–1001, July 2003.

[7] A. Cataldo, "Control algorithms for soft walls," 2004.

[8] E. Frew and R. Sengupta, "Obstacle avoidance with sensor uncertainty for small unmanned aircraft," in *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, vol. 1, pp. 614–619 Vol.1, Dec. 2004.

[9] J. K. Hedrick, B. Basso, J. Love, A. R. Girard, and A. T. Klesh, "Control of mobile sensor networks: A state-of-the-art review," October 2008.

[10] S. Prajna and A. Rantzer, "Convex programs for temporal verification of nonlinear dynamical systems," *SIAM Journal on Control and Optimization*, vol. 46, pp. 999–1021, 2007.

[11] A. Rantzer, "A dual to lyapunov's stability theorem," *Systems and Control Letters*, vol. 42, pp. 161–168, 2001.

[12] A. Rantzer and S. Prajna, "On analysis and synthesis of safe control laws," in *Proceedings of the Allerton Conference 2004*, 2004.

[13] T. A. Johansen, "Computation of lyapunov functions for smooth nonlinear systems using convex optimization," 1999.

[14] J. B. de Sousa, "Notes and concepts on the interpretation of hybrid systems," April 2008.

[15] K. H. Johansson, "Hybrid and embedded control systems lectures," January - March 2008.

[16] M. do Rosário de Pinho and M. M. Ferreira, "Análise matemática 3 - equações diferenciais," August 2004.

[17] V. K. Balakrishnan, *Introductory discrete mathematics*. Dover, 1996.

[18] J. P. Hespanha, "Hybrid control and switched systems lectures," September 2005.

[19] F. Blanchini, "Set invariance in control," *Automatica*, vol. 35, pp. 1747–1767, November 1999.

[20] R. Vidal, S. Schaffert, J. Lygeros, and S. Sastry, "Controlled invariance of discrete time systems," in *In Hybrid Systems: Computation and Control*, pp. 437–450, Springer Verlag, 1999.

[21] M. do Rosário de Pinho and M. M. Ferreira, "Análise matemática 2 - apontamentos das aulas teóricas," February 2008.

[22] D.-Z. Du and F. Hwang, *Computing in euclidian geometry*. World Scientific Publishing, 1992.

[23] R. Diestel, *Graph theory*. Springer, 2005.

[24] J. O'Rourke, *Computational Geometry in C*. New York, NY, USA: Cambridge University Press, 1998.

[25] H. Wang, J. Kearney, and K. Atkinson, "Robust and efficient computation of the closest point on a spline curve," in *Curve and Surface Design*, 2003.

[26] A. C. C. de Matos, "Apontamentos de análise numérica," September 2005.