MONITORAMENTO DE MÚLTIPLOS PERÍMETROS DINÂMICOS COM ROBÔS MÓVEIS

VÍTOR MACHADO GUILHERME BARROS

MULTIPLE DYNAMIC PERIMETER SURVEILLANCE WITH MOBILE ROBOTS

Dissertation presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Douglas Guimarães Macharet

Belo Horizonte

March 2020

Ficha catalográfica elaborada pela bibliotecária Belkiz Inez Rezende Costa
CRB 6ª Região nº 1510

Barros, Vítor Machado Guilherme.

B277m Monitoramento de múltiplos perímetros dinâmicos com robôs móveis / Vítor Machado Guilherme Barros — Belo Horizonte, 2020.
xxi,97 f. il.; 29 cm.

Dissertação (mestrado) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação

Orientador: Douglas Guimarães Macharet

1. Computação – Teses. 2. Robôs móveis – Teses. 3. Monitoramento de perímetros – Teses. 4. Rastreamento de trajetórias – Teses. 5. Cooperação de robôs – Teses. I. Orientador. II. Título.

CDU 519.6*82.10 (043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Monitoramento de Múltiplos Perímetros Dinâmicos com Robôs Móveis

VÍTOR MACHADO GUILHERME BARROS

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

Douglas Guimarães Macharet

PROF. DOUGLAS GUIMARÃES MACHARET - Orientador
Departamento de Ciência da Computação - UFMG

PROF. LUIZ CHAIMOWICZ
Departamento de Ciência da Computação - UFMG

Vinicius Mariano

DR. VINICIUS MARIANO GONÇALVES
Departamento de Engenharia Elétrica da Universidade Federal de Minas Gerais. -
Universidade Federal de Minas Gerais

Belo Horizonte, 30 de Março de 2020.

A princípio, nunca pensei que chegaria a escrever uma dedicatória para essa dissertação. Amigos, família... dar nomes é sempre a oportunidade de esquecer alguém – não por desmazelo, ou por falta de consideração, mas pelo simples esquecer. Ainda assim, quando me vi na necessidade de dedicar uma parte dessa dissertação a um órgão financeiro, não quis deixar de evocar alguns nomes aqui, e o motivo é simples: o apoio deles não foi o financeiro, não foi o intelectual, mas sim o pessoal; e isso nenhuma agência é capaz de dar a você. Sigo, pois, parcialmente o conselho de E. Hemingway: Write drunk, edit sober – parcialmente pois não pretendo voltar aqui.

Começo por você, Camila. Obrigado pelo apoio. Quando a conheci eu era apenas mais um formado que pleiteava uma vaga no mestrado e flertava com a decisão de partir para uma segunda graduação em medicina. Em você, estudante de medicina e hoje médica com todas as honras possíveis, eu via algo que sempre faltou em minha profissão: humanidade. É fácil lidar com robôs, máquinas, computadores. É fácil que eles façam o que você quer. Lidar com gente, por outro lado, já não é a tarefa que eu tenha alguma aptidão. E você me mostrou o lado bonito e humano de ser um ser humano para outro ser humano. A vida como eu entendia – em meio a fórmulas, teoremas, corolários... – é outra desde você. Enfim, você me acompanhou até o quase-fim dessa jornada, e por razões que o mundo colocou diante de você o que nos unia nos separou. Saiba que essa caminhada sem você não seria a mesma. De todos os caminhos, eu tomei aquele que foi certamente o mais bonito. E sem você nada aqui seria tão único, simples e singelo na sua mais simples forma. E definitivamente não seria tão cheio de surpresas. Ainda que após tantos encontros decidimos nos desencontrar, eu escrevo aqui com a certeza de que cada momento vivido e cada parte desse todo tem um pedaço seu. E por isso eu serei eternamente grato. Sem muito o que falar, ou como falar: muito obrigado. Por tudo.

Em seguida passo para você, Popola. Irmã de sangue. A pessoa que nutre a minha mais profunda admiração. A pessoa que eu sempre desejei ser: destemida, leal, e sonhadora. A pessoa que olhava para o desdém alheio quanto a suas ambições e ria, pois sabia que a sua risada ecoaria no seu sucesso. Você, cara irmã, é um espelho para o mundo. Quiserá o mundo ser bonito quanto você é. Ainda que seu muitas vezes chulo vocabulário e a sua voz alta por vezes me irritaram em momentos da mais necessária concentração, tenho para mim a certeza de que ainda há de existir alguém com a coragem de ser o primeiro pássaro a voar frente aos arqueiros como você tem. Você é a mais bonita representação de coragem que eu tive a oportunidade de conhecer em minha vida – e sou eternamente grato pela vida ter nos unido por irmandade, e principalmente respeito.

Não poderia me esquecer daqueles que poderiam ser a referência à magnum opus de Dumas: meus três amigos, meus três irmãos que o sangue não é compartilhado: Renato, Matheus e Tiago. Cada um de vocês foi de particular importância em um momento da minha jornada pelo mestrado. Arriscar dizer que um foi mais importante que o outro seria dizer a Dumas que os três mosqueteiros não formavam um tripé, mas alguns poucos pares conjugados. Cada um de vocês, à sua maneira, e no seu tempo, foi fundamental para que essas páginas que se seguem tenham algum sentido, alguma conclusão, e certamente um fim. A importância de vocês para mim e para esse trabalho – mesmo que nenhum de vocês saiba propriamente ligar um computador – transcende a intelectualidade por trás do que aqui está escrito. São anos nas trincheiras, caros amigos, e mais importante do que a guerra é quem está ao seu lado. Obrigado por me acompanharem nessa batalha.

Aos meus grandes irmãos da "Confraria", dos "Sócios" e do "Bonde da (insira nome bizarro aqui)": Victor Lima, Magno, João Victor Fagundes, Breno, Pedro, Samuel, Rafael, Guilherme Maia, Guilherme Borges, Guilherme Henrique, Victor Demétrius e João Victor Guerra: existem poucos os que têm a sorte de encontrar amigos como vocês pelo mundo. Eu sou um deles. E aqui expresso a minha imensa gratidão por poder falar de vocês e lembrar da palavra amigo. Saibam que todos vocês – à sua maneira – detém o mais belo sentimento de amizade que posso nutrir. Obrigado.

Aos meus grandes amigos de graduação, em seguida mestrado e agora também de trabalho: Rafael e Gabriel, o que seria de mim sem vocês? O que seria de mim sem o "Refs do Gnobre"? Vocês tornam o meu dia a dia mais leve, minhas amarguras mais amenas, e meus risos da manhã até a noite algo mais feliz. Saber que mesmo após a tortura da graduação e os infortúnios da pós, nós continuamos de pé, é algo que me diz que mantive minha cabeça no lugar, mesmo que ela tenha navegado bastante antes de chegar ao cais. Vocês mostram o que o crivo de uma boa seleção na graduação faz com profissionais: competentes, honestos, leais, e acima de tudo, parceiros. Para vocês, dedico toda a minha admiração, e espero algum dia os alcançar na corrida.

Jacy. Sim, você, mãe. Poucos são os privilegiados de ter uma mãe como você. Eu nunca vou entender os motivos pelos quais eu tive esse privilégio, e também já não é de minha alçada buscar razões ou motivos. Você, mãe, que sempre esteve ao meu lado, principalmente nas muitas tristezas que carrego desde aquele fatídico primeiro diagnóstico, tem um papel que muitas vezes eu me esqueço na minha vida. Ainda que eu seja ingrato a tudo que você fez e faz por mim, gostaria apenas de dizer que por ti tenho um amor incondicional. Seu apoio, mesmo quando eu parecia navegar sem rumo, sem certeza, sem razão para terminar esse trabalho ou iniciar uma nova carreira, é responsável pela pessoa que sou hoje. E a cada dia que passa, tenho mais e

mais certeza de que o seu trabalho foi árduo, e que um dia eu ainda hei de o honrar.

Paulo. Pai. Sei que o mestrado nunca foi algo que você desejou para mim. Sei que no momento em que eu arrumei um emprego, migrei da situação de bolsista para a situação de assalariado, você comemorou – não em minha frente, mas tenho certeza que internamente. Sei que eu não chegaria aqui se não fosse aquela breve briga no sinal, indo para a escola, onde eu dizia que gostaria de cursar algo diferente de engenharia, e que você disse que se eu era bom com números então meu lugar era na engenharia. Por mais tortuosos que tenham sido nossas decisões e caminhos, por mais estranha que tenha sido a forma como você me colocou em um caminho que eu jamais quis trilhar, cá estou. Já que sou, o jeito é ser (C. Lispector).

Jenecy. Tia. A primeira mestra da família onde a graduação já parecia um título extremamente avançado. A pessoa que me mostrou os primeiros passos da academia. A pessoa que hoje compartilha comigo as dificuldades do seu tempo, onde o ápice da vitória era não ter que datilografar outra página inteira corrigida por seu orientador, enquanto eu, aqui, edito linhas em tempo real. Obrigado pelo apoio nessa jornada acadêmica. Obrigado por ter sido você a primeira pessoa a dizer que o mestrado não seria apenas mais um curso, mas uma exigência como eu nunca tiverá antes. Talvez tivesse eu tomado esse conselho de forma mais séria, estaria escrevendo menos melancolicamente. Ainda assim, obrigado por todo o carinho.

No rol de pessoas que eu pensei que jamais escreveria sobre, dedico – antes da agência – esse último agradecimento a você, Douglas. Você que foi meu orientador, que suportou minhas discordâncias, que nos últimos meses trocou comigo e-mails onde parecíamos travar uma briga sem fim. Enfim, o fim. Ainda assim, cabe a mim agradecer de maneira sincera pelos ensinamentos dados. Ainda que eu não tenha chegado ao fim da maneira como você e eu esperávamos, conheço nossa trajetória, conheço os meus erros, defeitos e principalmente minhas vaidades e arrogâncias, e por isso admito que qualquer um em sua posição já teria desistido. Obrigado por não desistir.

Sobre perseguir algo tão intensamente como eu persegui esse título, nunca encontrei palavras mais representativas que a de David Foster Wallace, e por isso as coloco aqui: It now lately sometimes seemed a black miracle to me that people could actually care deeply about a subject or pursuit, and could go on caring this way for years on end. Could dedicate their entire lives to it. It seemed admirable and at the same time pathetic. We are all dying to give our lives away to something, maybe. – David Foster Wallace.

Acknowledgments

This work was partially financed by Coordenação de Aperfeiçoameto de Pessoal de Nível Superior (CAPES) – FinanceCode 001, Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and Fundaçnao de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG).

"Oh me! Oh life! of the questions of these recurring,
Of the endless trains of the faithless, of cities fill'd with the foolish,
Of myself forever reproaching myself, (for who more foolish than I, and who more faithless?)
Of eyes that vainly crave the light, of the objects mean, of the struggle ever renew'd,
Of the poor results of all, of the plodding and sordid crowds I see around me,
Of the empty and useless years of the rest, with the rest me intertwined,
The question, O me! so sad, recurring—What good amid these, O me, O life?

Answer.

That you are here—that life exists and identity,
That the powerful play goes on, and you may contribute a verse."
(Walt Whitman)

Abstract

The usage of mobile robots to perform tasks that were previously done only by stationary objects or humans has been one of the main focus of studies nowadays. Activities such as patrolling, surveillance, rescuing, tracking, mapping and many others, are examples of tasks in which using robots can be a determinant factor to improve quality and robustness of one solution.

In this work, perimeter patrolling activities are addressed in a way that a team of robots can be used to perform such activity in fully observable and time varying environments and cooperating to maintain as many regions covered as possible. This problem is addressed by decomposing it in three sub problems: (i) determining how to cover each region, by defining a perimeter around it; (ii) establishing a communication pattern between the agents in a way that surveillance missions can be communicated without the necessity of a centralized command center that will determine which robot will cover which area; (iii) guiding the robots to their patrolling mission in a organized way so that they can spend most of their time executing the patrolling task instead of navigating to its target.

Simulations and mathematical analysis using multiple robots in different environments were performed to illustrate the approach adopted to effectively follow multiple dynamic perimeters.

Keywords: Time-Varying Curves, Navigation, Trajectory Tracking, Patrolling, Robot cooperation.

Resumo

O uso de robôs móveis para execução de tarefas previamente executadas apenas por objetos estacionários ou humanos é um dos principais tópicos de desenvolvimento de estudos e pesquisas atualmente. Atividades como o monitoramento de regiões, vigilância, operações de resgate, rastreamento de trajetórias, mapeamento e muitas outras, são exemplos de tarefas nas quais o emprego de robôs pode ser um fator determinante para uma melhor solução.

No presente trabalho, tarefas de monitoramento de perímetros são estudadas de forma que equipes de robôs possam ser empregadas na realização da mesma em ambientes dinâmicos e completamente observáveis de forma que tais equipes cooperem para manter o maior número de regiões possível sob vigilância. Nesse trabalho, esse problema é trabalhado na divisão do mesmo em três subproblemas: (i) determinar como cobrir cada região, definindo um perímetro ao redor da mesma; (ii) estabelecer um padrão de comunicação entre os agentes de forma que as tarefas possam ser atribuídas sem a necessidade de um sistema central de comando que determina qual(is) robô(s) devem cobrir quais regiões; (iii) guiar o(s) robô(s) até a região de sua missão de maneira organizada, de forma que eles possam a executar durante o maior tempo possível.

Simulações e análise matemática utilizando múltiplos robôs em diferentes cenários foram realizadas para ilustrar a abordagem adotada para executar a tarefa de monitoramento de múltiplos perímetros dinâmicos de maneira eficiente.

Keywords: Curvas dinâmicas, Navegação, Rastreamento de Trajetórias, Vigilância e Patrulhamento, Cooperação de robôs.

List of Figures

List of Tables

List of Algorithms

List of Symbols

$\boldsymbol{\mathcal{P}_k}$	Set k of unique points that represents the boundary of a perimeter
$\Omega_{\boldsymbol{\mathcal{P}_k},\mathbf{t}_i}$	Perimeter defined by the set of points $\boldsymbol{\mathcal{P}_k}$ in a given time $\mathbf{t}_i$.
$\mathcal{B}(\Omega_{\boldsymbol{\mathcal{P}_k},\mathbf{t}_i})$	*Basis Spline* interpolation of the perimeter delimited by $\Omega_{\boldsymbol{\mathcal{P}_k},\mathbf{t}_i}$.
$\boldsymbol{\mathcal{B}_{all}}$	Set containing all Basis Splines.
$\boldsymbol{\mathcal{R}_{all}}$	Set containing all robots available for patrolling tasks.
$\boldsymbol{\mathcal{F}_A}$	Attraction force between a robot and its goal.
$\boldsymbol{\mathcal{F}_R}$	Repel force between two robots.
$\boldsymbol{\mathcal{F}_{res}}$	Resultant force. Vector sum of of $\boldsymbol{\mathcal{F}_R}$ and $\boldsymbol{\mathcal{F}_A}$.
C^n	Continuity of degree n of a function.
s_k	Sample k of the set of points of a Perimeter.
δ	Threshold value for the distance between two samples of the same perimeter.
x	State vector of a robot ($[\mathrm{x}, \mathrm{y}, \theta]^T$).
$\|\dot{\mathrm{x}}(\mathbf{t}_k)\|$	Magnitude of the first derivative of the state vector of a robot in a given time $\mathbf{t}_k$.
$\mathcal{U}$	Utility value of performing a patrolling task.
C_e	Coverage efficiency.
L	Perimeter size.
E_{ct}	Estimated coverage time.
ct	Coverage time.
v_{max}	Maximum velocity of a robot.
t_t	Traveling time of a robot to its target.
x_c	Position reference of the robot on the x-axis.
y_c	Position reference of the robot on the y-axis.

θ_c	Angle between the vehicle's velocity vector and the x axis.
v_c	Linear velocity control input.
ω_c	Angular velocity control input.
$\vec{e}$	State tracking error vector.
u_1	First input signal to the control loop.
u_2	Second input signal to the control loop.
v_d	Feed-forward linear velocity control signal.
ω_d	Feed-forward angular velocity control signal.
ω_n	Natural angular frequency.
ξ	Damping coefficient.
$V(\cdot)$	Candidate Lyapunov function.

Contents

Chapter 1

Introduction

Nowadays, it is common to use robots to perform activities that are potentially risky for human beings. Some of these activities are related to rescue missions and damage control in environmental disasters such as wildfire, oil spills, radiation leaks, among others. When such situations happen, a quick and effective response is necessary to prevent more or worse damages, and to secure life of living organisms in the disaster's area.

As an example, considering environmental disaster scenarios, it is possible to identify a common characteristic in many of them: the existence of a *perimeter* that surrounds the affected areas, which can be (i) one or multiple, (ii) dynamic or static. As an example, a wildfire can start in a single or multiple regions of a given area and spread over time. Even when considering time-varying regions or anomalies, it is possible to determine perimeters that work as boundaries on the regions and anticipate the dynamics of these phenomena even when a dynamic model for the phenomenon is not available. Once those boundaries are defined, it becomes feasible to deploy teams of robots to perform perimeter surveillance missions which can be extremely useful to support evacuations, or to dissipate the hazard.

Many real-life situations can be modeled as a dynamic perimeter surveillance problem like forest fires, oil spills, and radiation leaks. All those situations share a common characteristic which is the existence of perimeters that determine the region containing those hazards and must be kept under surveillance. Also, those regions have a dynamic behavior, which means that the boundaries of those perimeters can shrink or expand over time.

1.1 Contextualization

As technology expands and become every day more and more a part of our daily activities, the study of robotics is progressively growing, especially when it is associated with many other technologies and studies in a great variety of applied sciences areas, such as artificial intelligence, control, computer vision, machine learning, engineering, and many others.

One of the main applications of robotics is the development of agents that can interact with humans to help them to accomplish a given task. The first robots were mainly developed for industrial purposes, in which they were used in assembly lines to perform repetitive tasks accurately, making technologies such as portable telephones, microchips, and many others possible [Siegwart and Nourbakhsh, 2004]. Although extremely precise and useful, those agents - many times refereed to as manipulators or robotic arms - suffered from a lack of mobility, which restrained their capability to the area they were designed for.

One solution to this problem was to generalize the previously stationary object into a mobile platform, to make it reach different places, and add new features, such as cameras, sensors, and a group of other different tools, so they could perform a whole new set of tasks. Some of those tasks are still subject of many researches such as safely mapping an unknown region [Wolf and Sukhatme, 2004; Stachniss and Burgard, 2005; Walcott-Bryant et al., 2012]; environment reconnaissance and surveillance tasks [Kingston et al., 2008; Acevedo et al., 2013; Saldaña et al., 2015; Jahn et al., 2017]; detection and tracking of persons, objects and different sort of items [Wang and Thorpe, 2002; Bellotto and Hu, 2009; Jafari et al., 2014]; perform search and rescue missions in case of catastrophic events [Shiroma et al., 2005; Ruangpayoongsak et al., 2005; Calisi et al., 2007]; and many other activities that could not be previously performed due to the lack of mobility of the robots.

Still, just provide mobility and attach advanced tools and gadgets to robots is not the ultimate solution. It is extremely necessary to make those robots move efficiently so they could use their new gadgets effectively. Due to this fact, when it comes to mobile robots it is extremely important to study how their movement can be determined and combined to optimize one or more intended tasks. It is of utmost importance to consider energy-saving [Kim and Kim, 2014; Tokekar et al., 2014], minimize the navigation time [Kim and Kim, 2011], optimize a robot's team task performance [Ding et al., 2010; Altché et al., 2016] and other activities that can use one or multiple robots.

One of the tasks that have been the focus of many studies is related to the use of mobile robots - usually coordinated teams of them - to perform surveillance and

patrolling missions. Most of those missions' main objective is to maintain a specific area under protection, even though the area under surveillance is static or dynamic.

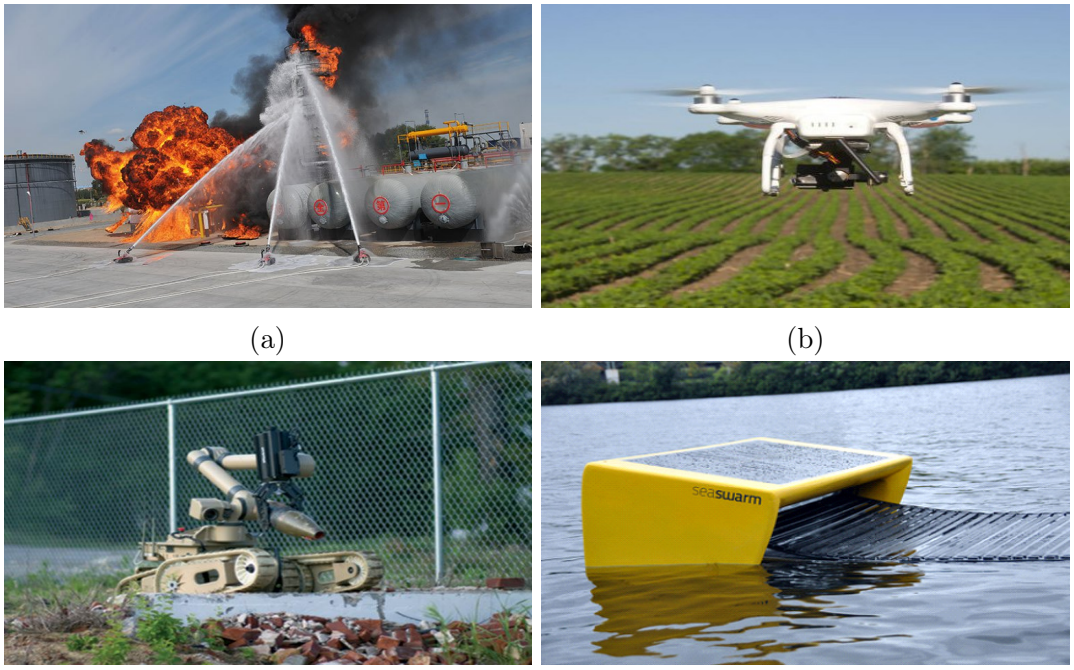
In this thesis, the main focus is to use teams of robots to perform perimeter surveillance on multiple areas defined by samples that represent coordinates of the intended perimeter. The presented solution for such a problem is based on the usage of mathematical interpolation of those samples to define the contour of the perimeter to be under surveillance. Different from what has been presented in the literature so far, we also provide a solution that considers that those samples are capable of moving over time in a random pattern, granting a dynamic behavior to the perimeter under surveillance. To deal with this problem, we also propose techniques to merge and split perimeters in case samples that belong to different perimeters start to get closer to each other narrowing the path that a vehicle can safely pass, or drift away from each other creating passages that are safe for the vehicles to pass through.

1.2 Motivation

Many real-life situations can be modeled as a dynamic perimeter surveillance problem. Considering the usage of robots, security activities such as perimeter surveillance are of real interest in research and real-life applications. Those tasks consist in using a vehicle or a team of vehicles to patrol a given perimeter. It is possible to identify a common characteristic in many of perimeter surveillance tasks: the existence of a *perimeter* that surrounds the affected areas, which can be (i) one or multiple, (ii) dynamic or static. Also in many cases, those regions have a dynamic behavior, which means that the boundaries of those perimeters can shrink or expand over time.

In this context, perimeter surveillance can be used to describe many situations such as monitoring oil spills, which require extreme precision to continuously deal with the spill spread; forest fires, in which the usage of robots can minimize the risk to humans; escorting groups moving around an unknown environment, which can help those groups to move safely in such environment; and also be used to perform a gas leak detection in a defined perimeter, which can help to secure residential and industrial zones that use gas as a power source.

(a): `https://apfmag.mdmpublishing.com/design-development-and-application-of-firefighting-robot-platforms/`
(b): `https://dairynow.ca/crop-surveillance-drones-take-your-farming-to-next-level/`
(c): `https://www.popularmechanics.com/technology/robots/g795/3-robots-that-braved-fukushima-7223185/`
(d): `https://sap.mit.edu/article/standard/swarm-robots-clean-oil-spills`



(a) (b)

(c) (d)

Figure 1.1: (a) Robots used to extinguish fire on a industrial complex. (b) Crop surveillance drone used in a farm. (c) Robot navigating through Fukushima after the radiation leak. (d) Robots being used to clean up oil spill.

It is also really important to notice that using autonomous vehicles to perform surveillance missions is considered a really useful approach as those tasks usually expose humans to risky activities and sometimes require precision beyond human reach. Therefore, it is possible to identify two immediate gains considering the usage of autonomous vehicles: (i) the security of the people involved in the task, (ii) and the increase of the overall performance of the task given the precision of the robot involved in it.

Even when considering time-varying regions or anomalies, it is possible to determine perimeters that work as boundaries on the regions and anticipate the dynamics of these phenomena even when a dynamic model for the phenomenon is not available. By doing that, it becomes feasible to deploy teams of robots to perform perimeter surveillance missions which can be extremely useful to support evacuations, or to dissipate the hazard.

1.3 Problem Statement

In this thesis, our main focus is to deal with the dynamic perimeter surveillance problem. This problem can be seen as defining the boundary of a region defined by the contour produced by a set of unique and identifiable planar points which is enclosed by a well-delineated perimeter.

In this thesis, the dynamic perimeter surveillance problem is divided into three key problems, as shown in Figure 1.2.

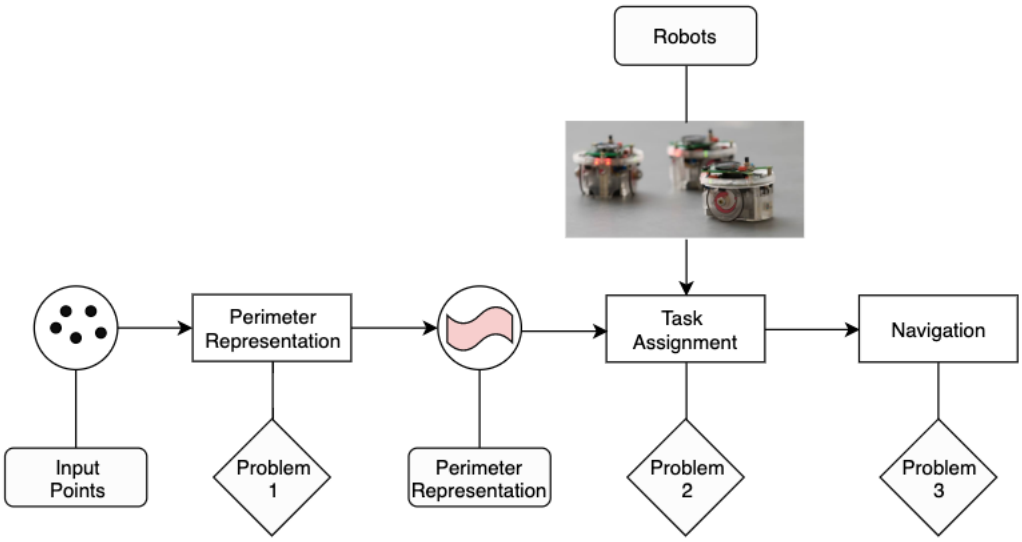
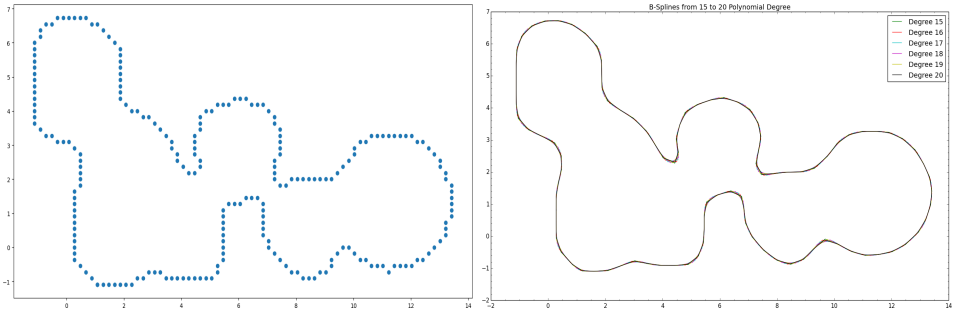


Figure 1.2: Pipeline describing the tasks involved in this work.

The first problem is how to represent a perimeter using only a set of samples which its only information is its position, and how to smoothly modify this representation as the samples randomly move over time. We address this problem as a dynamic perimeter representation. One can formally describe a dynamic perimeter – on the context of this thesis – as the boundary of a region defined by the contour produced by a set of unique and identifiable planar points $\mathbf{P}$ which is enclosed by a well-delineated perimeter by using a *Basis Spline* interpolation. This perimeter is a time-varying region indexed by its set of points ($\mathbf{P}$) and time ($\mathbf{t}$, $\mathbf{t} \in \mathbb{N}^0$).

Once defined what is and how to represent such perimeters, our second key problem is related to use robots to navigate to those areas and keep navigating on their boundaries as the samples move over time and the perimeter is modified. In this thesis, we address this problem as a surveillance task. A surveillance task is the task of allocating one or more robots to an area and cover its perimeter by continuously navigating as close as possible to its boundary. At this point, it is important to notice

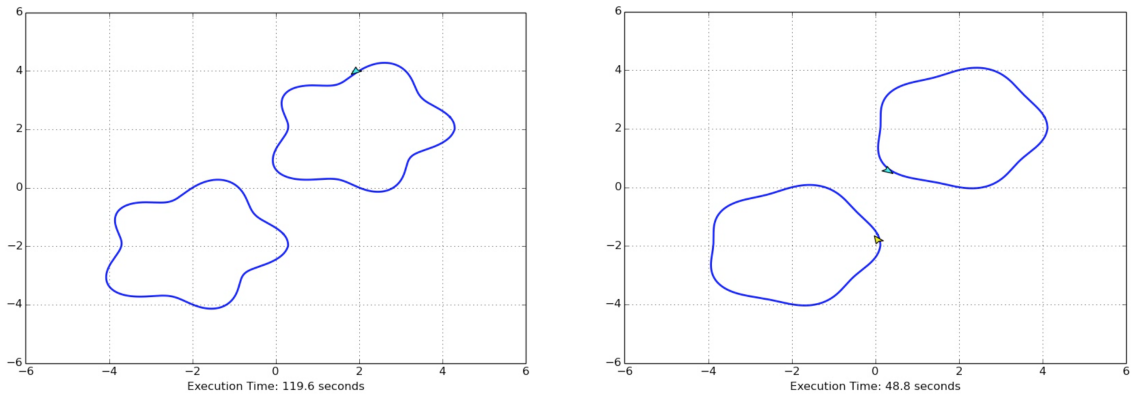


(a) (b)

Figure 1.3: (a) Points that define a perimeter. (b) Basis Spline using multiple interpolation degrees to interpolate the planar points. Each interpolation level is represented by one color.

that one robot is capable of tracking one and only one area at time even if more than one area is in the field of view of the robot.

Finally, our third problem is related to the allocation and coordination of those robots so they can perform the dynamic perimeter surveillance task. Considering scenarios with multiple robots and/or multiple perimeters, in this thesis we aim to create a solution in which we maximize the covered area regardless of using one or multiple robots. We propose a greedy strategy for allocating the robots to their areas which minimizes the time to reach the perimeter to be covered to stay navigating on its boundary for a longer time. We also consider scenarios in which the robot moves from one perimeter to another to also maximize the number of visited areas instead of visiting just one of them exhaustively.



(a) (b)

Figure 1.4: (a) One robot – represented by the blue triangle on the north-most perimeter – performing surveillance on one of the two perimeters. (b) Two robots – represented by the yellow and blue triangles – performing surveillance on two different perimeters.

As expected, handling all those problems at once would not be possible if some

assumptions were not adopted. In this thesis, we assume that the position of the samples that define one or multiple perimeters are known with an error of just 1%. We also defined that all of our robots are equal and described as a differential robot with a sensor capable of collecting local information in a discrete-time. In addition, the robot start its navigation considering the existence of M ($M \geq 1$) sets of planar points, used to define M different perimeters at time $\mathbf{t} = 0$, meaning that each set of points is already clustered and will be used to determine its perimeter. The robot updates its sets as it navigates and sample the environment. Regarding the properties of the dynamic environment, we consider a relatively slow-moving perimeter as the robots move much faster than the perimeter. This is an important assumption because robots have speed limitations and tracking a completely arbitrary perimeter is not always possible as it can expand or collapses faster than the robot can track. Finally, it is important to mention that another important assumption is that the vehicles are capable of communicating with each other at any given time, which is commonly refereed in the literature as global communication; and are also aware of the position of each other and the samples at any given time, which is commonly refereed in the literature as a fully observable environment.

Considering all those assumptions and definitions one can summarize it as a problem in which given a group containing N robots and having M time-varying perimeters to be patrolled, our main objective is to maximize the total area under surveillance over time.

1.3.1 Objectives

Since most of the real-life situations do not attend to the particular restrictions mentioned before, and it is common to deal with the patrolling tasks on stochastic environments and with a limited number of resources (e.g. a limited number of robots, multiple regions to put under surveillance, and others), it is necessary to study this problem tackling real-life problems and issues.

In this work, we handle four different scenarios that involve the usage of single or multiple robots covering one or multiple perimeters. Our objectives regarding those four different scenarios are:

1. Focus on covering as many perimeters as we are capable of.
2. Minimize the time to target – where we consider the target as the perimeter to be under surveillance.
3. Provide a technique that is flexible when it comes to the perimeter shape.

With those objectives in mind, we consider that it is more advantageous to cover multiple perimeters at the same time than to cover more areas of a specific one. This characterizes our covering strategy as a greedy one. Also, by minimizing the time to target, our strategy aims to keep robots spending most of their time doing patrolling tasks instead of navigating to a perimeter. Finally, it is important that different perimeters shapes can be covered, as the environments are stochastic and there is no information on its dynamics.

In order to do it is considered in this work that the environment in which the robots are performing patrolling tasks is dynamic, which, in this case, means that the regions under surveillance will change over time. Those regions will be delimited by points, and the interpolation of those points will determine the boundaries of the region. Other considerations regarding the approach addressed in this work are related to a full knowledge by the agents regarding their pose during all the navigation and their objective position once assigned a region of interest for them to patrol. Robots are also able to distinguish other robots one from another, basically meaning that they know what each agent is.

1.4 Contributions

As shown on Figure 1.2, this thesis will approach three different problems: perimeter representation once a set of input points is given; task assignment, in which a task is described as performing a dynamic perimeter surveillance on the perimeters previously calculated; and finally, as the task are allocated, perform a navigation on those perimeters. Given those three problems, one can define that this thesis contribution can be described as:

- A mathematical approach that is used to define a dynamic perimeter using a set of points.
- Two algorithms to perform merge and split of regions defined by sets of points as they approach or drift apart from each other.
- A control strategy based on tangent linearization that is used to make the robots converge to and navigate on a designated perimeter.

Summing up, this thesis proposes a framework in which our perimeters are represented by *B-Splines* and the trajectory tracking control design is based on tangent linearization along the reference trajectory along with a potential fields technique to

avoid collisions. In this work, the main focus relies on maximizing the coverage of multiple perimeters, as by covering multiple perimeters it is possible to obtain more information about that region and perform a better surveillance task.

1.5 Thesis Roadmap

The remainder of this work is organized as follows: Chapter 2 presents a brief review of literature, analyzing trajectory tracking methods, trajectory convergence and following, and multi-agent task assignment. In Chapter 3, we present our methodology and the basic concepts for a better understanding of the proposed approach to deal with the dynamic trajectory tracking problem. Chapter 4 presents the experiments used to validate the proposed methodology. Finally, Chapter 5 present the final remarks, considerations and future works.

Chapter 2

Related Work

Motion planning is one of the basic problems in robotics which can be defined as the attempt to move a robot from a given initial state to a destination state while avoiding static or dynamic obstacles that may be on its way. Choset et al. [2005] defines the difference between *path* and *trajectory* as: a *path* is a sequence of positions, defined in the robot configuration workspace, which is the space of all positions or configurations that the robot can achieve. Therefore, a *trajectory* is a path with a velocity profile along with it, defined in the higher dimensional state space where every point defines a position, or a configuration, and the velocity vector at that point. Hence, trajectory planning algorithms are usually used to solve dynamic problems, in which the robot must make decisions based on how it interacts with the environment; whilst path planning algorithms are usually used to solve geometric or kinematic problems, and mainly used before assigning the robot to a task.

Trajectory planning and tracking is one of the most important tasks that one or many mobile agents must perform to navigate through an unknown, partially known or completely known environment efficiently and safely so it can accomplish a given task. Some of those tasks are the escort of a group from an initial position to a goal position, the surveillance of a given region, the avoidance of a determined region, the convergence of the group to a specific region, and many others.

2.1 Trajectory Planning

When it comes to trajectory planning, two different approaches are considered: Offline Trajectory Planning and Online Trajectory Planning. Those two different approaches are widely studied in the literature, and their main difference relies on how the robot will plan its trajectory: before or during its navigation, respectively.

According to Shiller [2015], generally, it is desirable that the trajectory to the goal is computed online to allow a reactive approach when it comes to changes in the environment and measurement errors by the robot's measurement mechanisms such as its sensors. However, many difficulties regarding this approach are commonly faced such as the high dimensionality of the state space in which the robot must perform a search, the geometric nature of the obstacles, the cost function to be optimized, and the robot's kinematic and dynamic model. Those difficulties prevent many solutions to be computed sufficiently fast, making a purely online solution not feasible in real-world scenarios.

Due to the difficulty in computing online trajectories on unknown or partially known environments, online trajectory planners are commonly used in association with offline trajectory planners. This association results in a motion planner that uses offline planners as a global viewer of the environment, trying to select the optimal trajectory to the goal, whereas the online planner may select the next move based on a partial view of the environment, allowing the robots to perform a reactive action in face of any adversity not previously calculated by the offline planner.

2.1.1 Offline Trajectory Planners

According to Shiller [2015], offline planners have the advantage of producing globally optimal solutions if the environment is completely known, having its drawbacks mainly related to computational complexity, completeness, and optimality (local and global). Most applications that use offline planners are those related to repeatable tasks in static environments where optimality is essential.

Before start planning how to navigate through a region, especially if the environment is known, it is possible to represent the intended region if partial or total information regarding it is previously known. Many works tend to represent a region as a set of curves, being those curves the ones that must be avoided or tracked by the agent. Another common way to represent a region is by defining it as a set o key-points that must be covered during the movement of the agent.

Some trajectory representations relied on solutions in which previously known key-points of the trajectory are the only available information and the path is designed by connecting those key-points. This kind of approach is studied in Likhachev et al. [2005], in which a trajectory is represented by connecting only two key-points: an initial and a goal point. Those two points are connected while the robot moves towards the goal in an unknown and dynamic environment. When considering that this approach must perform a search in a 4D space state (the agent's position (x, y), its orientation

and its velocity) it is possible to understand that it can be extremely costly to the agent to perform such task given that a 4D space state is likely to have a great number of states that must be checked to select the best one. Also, since many times only partial information regarding the environment exists, some generated solutions using its initial information may turn out to be invalid or suboptimal as it receives updated information using its sensors. Hence, the authors propose an incremental re-planning approach that performs a series of searches using decreasing inflation factors to generate a series of solutions represented by a graph. When there are changes in the environment affecting the cost of some of the edges in the graph, locally affected states are placed in a priority queue. States on the queue are then processed until the current solution is guaranteed to be at least as good as a threshold value ϵ. Their algorithm, named Anytime Dynamic A* (AD*), is the combination of two algorithms, the Anytime Repairing A* (ARA*) algorithm, proposed in Likhachev et al. [2003] and the D* Lite algorithm, proposed in Koenig and Likhachev [2005]. An example of navigation using the AD* Algorithm is presented in Figure 2.1.

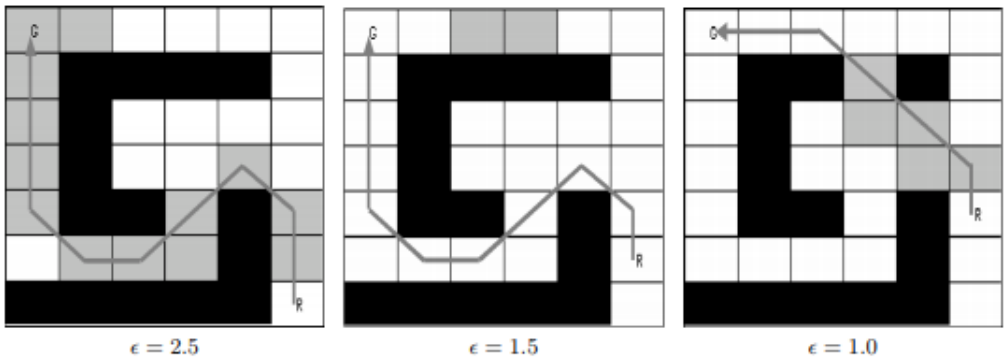


Figure 2.1: A simple robot navigation example using the AD* Algorithm with different threshold values as presented in [Likhachev et al., 2005].

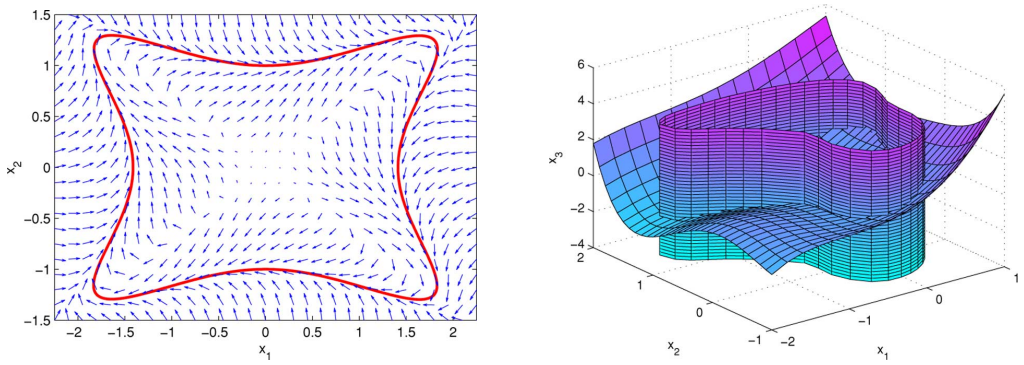
When not considering a graph approach, another way to represent a trajectory, studied in Chaimowicz et al. [2005], considers the usage of a swarm of robots to synthesize shapes and patterns, converging and spreading along 2D curves. Those curves are represented by specific implicit functions, that are generated by interpolating several user-selected constraint points. The implicit function $f(x, y)$ is produced by a weighted sum of radial basis functions (RBF) created by the interpolation of the constraint-points.

Following the same idea of using implicit functions, Goncalves et al. [2010], proposes the representation of a generic closed-curve $\tau(t)$ which may be static or time-varying in a n-dimensional space, such that by computing its vector field it is possible

to converge and circulate along it. The target curve $\tau(t)$ is expressed as the intersection of $n-1$ surfaces of co-dimension 1. As stated by the authors the set $\tau(t)$ is a set of all points that lie in the intersection of the levels sets $\alpha_i = 0$, in which:

$$\tau(t) = \{\alpha_i(x_1, x_2, ..., x_n, t) : \mathbb{R}^{n+1} \vdash \mathbb{R}, (\forall i \leq n-1)\}, \tag{2.1}$$

are functions with bounded second order derivative such that $\tau(t)$ is connected and one-dimensional $\forall t \geq 0$. An example of how a vector field approach can be used to determine closed curves for navigation is depicted Figure 2.2:

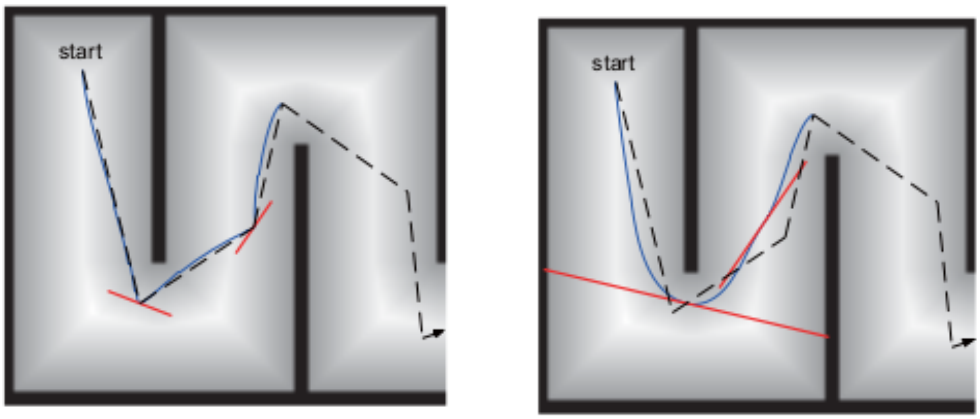


(a) A vector field (b) Intersection of two surfaces.

Figure 2.2: (a) A vector field and a curve, (b) two dimensional surfaces in $\mathbb{R}^3$ which the intersection defines a 1-D closed curve [Goncalves et al., 2010].

Different from the previous approaches, another representation technique, studied in Lau et al. [2009], consider the trajectory $\langle x, y \rangle = \hat{Q}(t)$ of an agent as its position on the ground in the world coordinates over time. Therefore, the trajectory shape is defined by a parametric curve $Q(u)$, in which its non-linear internal parameter u is mapped to the metric system s along the path via numerical arc-length parameterization. Hence, the curve $Q(u)$ defines the 2D shape of the trajectory, which consists of N_Q joined segments $Q_i(u_i)$, $(i \in [0, N_Q - 1])$. To connect those segments, the authors use quintic Bézier splines, matching two segments by setting both second and first-order derivative to the same value. Considering that the beginning of the trajectory is a series of positions without orientation that are found to be collision-free when connected by a straight line, their method iteratively optimizes the trajectory into a curved trajectory. Figure 2.3 presents how this approach works:

Another perspective is proposed by Ko et al. [2014]. The authors propose the use of an Artificial Vector Field which is generated by using an modified version of the RRT algorithm to describe the desired trajectory. The modification used by the authors considers a unit magnitude vector field $f(q)$, defined on some configuration



(a) (b)

Figure 2.3: (a) Trajectory planned before, and (b) after optimization. The original trajectory was generated with sparse waypoints taken from a straight-line path (dashed). Trajectories are shown in blue, and tangents in inner waypoints are depicted in red [Lau et al., 2009].

space, with the initial configuration q_{init} and a final configuration q_{final} both given. Then, given a random node q_{rand}, the nearest tree node from this random node - q_{near} - a unit vector ($\hat{v}_{rand}$) is calculated from q_{near} towards q_{rand} and three additional vectors are also calculated in order to build the trajectory:

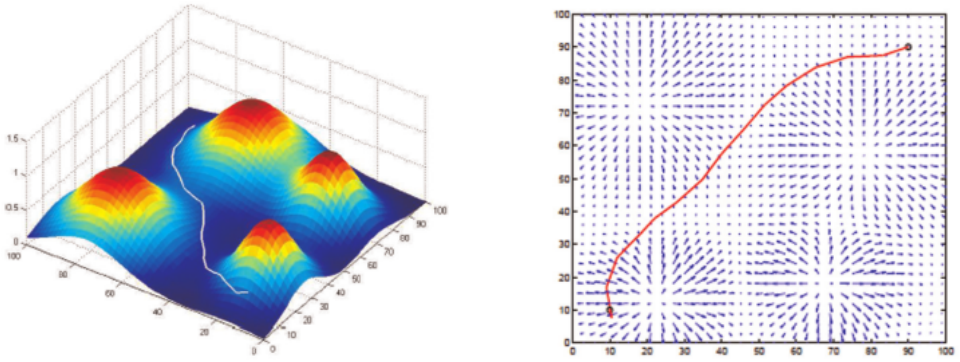
1. $\hat{v}_{field} = f(q_{near})$, the vector field at q_{near},
2. $\hat{v}_{new}$, the direction in which the tree extends,
3. $\hat{v}_{input}$, a unit vector lying on the plane spanned by $\hat{v}_{rand}$ and $\hat{v}_{field}$ such that for an appropriate choice for an scalar $\omega \in \mathbb{R}$ the equality

$$\hat{v}_{new} = \hat{v}_{field} + \omega \cdot \hat{v}_{input}, \tag{2.2}$$

holds.

Using a 3D workspace, Figure 2.4 presents the result of using this technique in order to find a path in which the potential value is strictly monotone decreasing and how this path can be represented in a vector field:

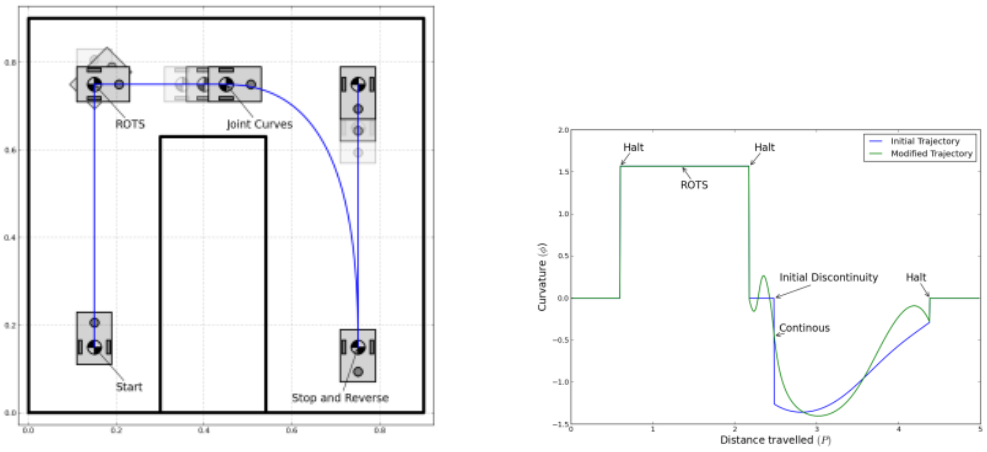
Aiming a purely generic representation solution for wheeled mobile robots navigation, Kjærgaard et al. [2014] adopts part of Lau et al. [2009] method to represent a trajectory and modifies it to produce trajectories that allow rotation-on-the-spot segments; considers that the input is a collision-free, possibly curved trajectory with pose information continuous in position and orientation; knots are curved-matched without



(a) (b)

Figure 2.4: (a) Minimum path in which the potential value is strictly monotone decreasing, and (b) the minimum path in vector field [Ko et al., 2014].

the need for the second derivative to be the same value; and allows knots to be non-continuous in curvature, in case a full-stop action is required. The final trajectory is also a curve. This approach is depicted in Figure 2.5.



(a) (b)

Figure 2.5: (a) The trajectory for a differential drive robot. (b) The representation of the original trajectory by third order Bézier segments and one rotation-on-the-spot segment. The curvature of the initial trajectory is illustrated as a blue curve and the green line in the figure illustrates the modified curvature of the trajectory [Kjærgaard et al., 2014].

2.1.2 *Online* Trajectory Planners

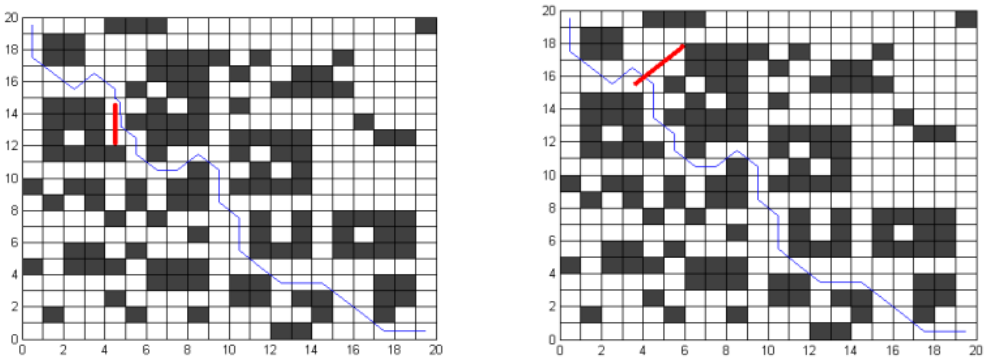
When considering that no information regarding the environment is available, trajectory representation is usually the outcome of a reactive approach, meaning that the

trajectory is built while the robot explores the environment. According to Shiller [2015], online planners are commonly used in applications where the target states are not previously determined, the environment is dynamic, the computation time required for a global solution delays the task execution, or the planners are just an alternative to a computationally expensive offline planner, even in static environments.

Kroger et al. [2006] proposes a trajectory planning algorithm in which the input parameters are completely arbitrary, meaning that the target position, the maximum velocity, acceleration, and jerk are neither considered as continuous nor constant. However, two restrictions must be taken into account: the velocity and acceleration in the goal position are supposed to be zero. In order to build the trajectory, the authors also consider that the trajectory is calculated in real-time during every control cycle, and due to the fact that the input values may change unpredictably, only the next sample point is calculated within one control cycle. Also, considering a n-dimensional space, where n is the number of degrees of freedom, the authors focus on mapping the output vectors of position, velocity and acceleration in a control cycle i ($\vec{p}_i$, $\vec{v}_i$ and $\vec{a}_i$, respectively) on the manipulator's *joint space* and the *Cartesian Space*. Thus, to perform such synchronization it is important that all n degrees of freedom reach their target position simultaneously at zero velocity and zero acceleration. The trajectory performed can be described as the setpoints for a lower-level control ($\vec{p}_i$, $\vec{v}_i$ and $\vec{a}_i$), and by performing geometrical and space-state transformations it is possible to represent the final trajectory as a curve.

Still, when dealing with unknown environments where static and dynamic obstacles are part of, it is hardly likely that the output of any algorithm is a curve that can be followed in any given situation. Since the environment contains dynamic obstacles, a position located in a previously calculated trajectory, might not be free due to the fact that one of the dynamic obstacles could have moved to this position. When considering that the obstacle motion is known, Dong-Shu and Hua-Fang [2011] proposes an algorithm that combines an Ant Colony Optimization (ACO) algorithm considering only the static obstacles in order to produce an optimal trajectory, and two strategies to avoid collision with moving obstacles. The first strategy consists in detect that when colliding *face-to-face* with the obstacle, the agent abandons its former path and process the construction of a new path considering the incoming obstacle as a temporary static object, and re-planning a local collision avoidance path - again using ACO - between its current position and the goal. A second strategy considers a *side-face* collision with the object. In this case, the robot stops in its current position, wait for the obstacle to move, and when the robot will no longer collide with the obstacle, it can move forward along the formerly planned path. This collision avoidance and navigation strategy is

presented in Figure 2.6:



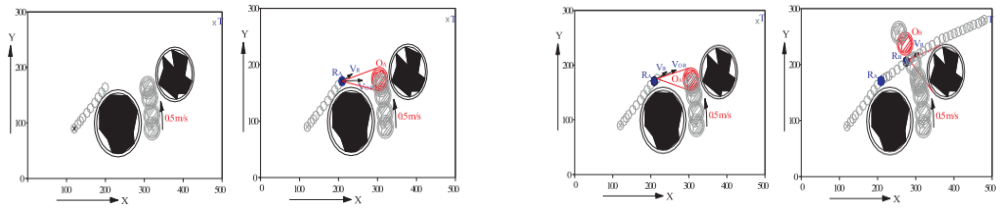
(a) Face to Face collision forecast (b) Side to Face collision forecast

Figure 2.6: The red thick line denotes the moving trajectory of dynamic obstacles, whose motion direction is from bottom to top (a) and from left bottom to right top (b). In both cases the behavior of the obstacle motion is known, thus re-planning the trajectory by can be done by the strategy adopted in [Dong-Shu and Hua-Fang, 2011].

When dealing with dynamic obstacles that have an unknown behavior, Raja and Pugazhenthi [2012] proposes a technique that despite the fact that the obstacles can assume any shape, be static or dynamic, and move in any direction and even perform rotations along their own axis, they are modeled as enclosing circles and amplified by a value to take care of physical dimensions of the robot and a safe maneuverable distance from the obstacles without collision. In order to proceed to its target, the agent checks (by using the data collected by its sensors) for any nearby obstacle. In case nothing is detected, the agent proceeds in a straight line towards the goal of satisfying its own kinematics and dynamics constraints. In case there is an incoming target, the authors use the velocity obstacle technique to prevent a collision. In case many obstacles are detected, a collision distance index is calculated to avoid the closest obstacle. To steer the agent out of any collision zone, most likely to happen when multiple obstacles are incoming, the algorithm proposed by the authors generates numerous collision-free combinations that satisfy the dynamics constraints, as can be seen in Figure 2.7. Then, a Particle Swarm Optimization technique is used to quickly select the best combination to have the shortest collision-free trajectory of the robot.

2.2 Trajectory Convergence and Following

Perform a navigation task is one of the most common tasks to be done when considering one or multiple agents. When considering that information related to the environment



(a) (b)

Figure 2.7: (a) demonstrates the robot moving tangentially to a statical obstacle and towards the goal followed by a prediction of collision with an incoming obstacle. (b) shows the robot steering out the obstacle cone predicted beforehand and reaching the target. The robot is represented by the blue polygon and the target is depicted in red [Raja and Pugazhenthi, 2012].

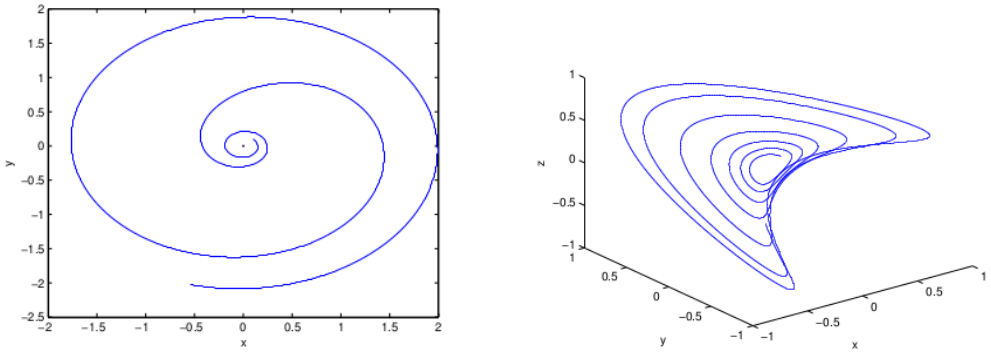
is available, trajectories end up being represented by curves, which can be expressed as implicit functions, splines and parametric curves (such as Bézier Curves), as a result of an offline representation of a trajectory. Even considering small deformations on the curve along the time, or some changes in the environment, some strategies regarding offline representation can be reactive to those changes.

Many techniques, such as the usage of potential fields, artificial vector fields, modifications on the RRT algorithm and others are used so a robot or a swarm of robots are capable to converge to the desired region and perform a series of tasks. One of the tasks is static positioning, in which the agent (or the swarm) will converge to a determined position (or a set of positions) in the curve and stay there. Another common task is to circumnavigate the target curve, where the agent or the swarm will not only converge to the curve but after reaching, the robot (or the swarm) will also circumnavigate it. Another common task is to avoid the described curve, consisting of exploring the entire space, except for the area delimited by the curve. All those activities are intimately related to surveillance, patrolling and guidance missions, usually the ones that a robot or a swarm is used for.

2.2.1 Single Agent Trajectory Following

When considering the usage of a single agent, Goncalves et al. [2009] proposes a technique based on Artificial Vector Fields for tracking a curve and circumnavigate it. On their work, the authors consider a closed-curve Γ, which may be static or time-varying, in a n-dimensional space such that its vector fields (that can also be static or time-varying) integral curves converge to and circumnavigate Γ. When it comes to tasks such as surveillance and monitoring an area by a single unmanned vehicle, the authors provide an efficient solution that deals with this problem considering time-varying

boundaries in n-dimensional space, hence terrestrial and aerial surveillance are issues that can be solved by the same technique just by modifying the boundaries dimensional space. As an example, depicted in Figure 2.8, dynamic curves can be defined in $\mathbb{R}^2$ for terrestrial surveillance whilst dynamic curves defined in $\mathbb{R}^3$ are able to perform aerial surveillance.



(a) (b)

Figure 2.8: (a) Convergence to a target curve with initial conditions $x_1(0) = 0$ and $x_2(0) = 0.1$. (b) Convergence to a target curve with initial conditions $x_1(0) = 0$, $x_2(0) = 0.1$ and $x_3(0) = 0.1$ [Goncalves et al., 2009].

In Jahn and Pimenta [2016], a combination of the VFRRT algorithm proposed by Ko et al. [2014] for trajectory planning, and Goncalves et al. [2009] technique for trajectory navigation based on Artificial Fields, results in a trajectory planning and tracking algorithm capable of finding a feasible path in a 3D workspace with static and known obstacles. The authors claim that a key advantage is that the sampling-based trajectory planner is dynamically and adaptively biased, based on an easy-to-construct vector field. Hence there is no preprocessing necessary to extend the vector field navigation to deal with static obstacles. Besides, even though two distinct techniques were used on their work, the good properties of both techniques were maintained: the rapid space exploration in case of obstacles provided by the VFRRT technique, and the quick convergence to a defined target curve provided by the Artificial Vector Field tracking technique. Still, the proposed solutions do not deal with time-varying vector fields and bigger than 3-dimensional trajectories.

2.2.2 Multiple Agents Trajectory Following

Considering the use of multiple agents, Chaimowicz et al. [2005] proposes a technique in which a swarm of robots converge to and spread along a 2D curve which is defined by an implicit equation. The curve can also be described as zero isocontour of a 3D

surface, whose values are negative for all points inside the boundary of the curve, positive for all points outside the boundary of the curve, and zero at the curve. On his work, it is used as a gradient descend technique in order to provide convergence to the robots to the desired position. Once the robots are close enough to the desired curve, repulsive forces will maintain them along the curve and they will be able to stay at the desired position, covering the shape of the curve, as shown in Figure 2.9. This work provides a decentralized methodology to control a swarm of robots to converge to and spread along a specific curve with a static shape. It is also important to notice that on this work the authors mention that an ideal relationship between the number of robots, the repulsive forces threshold to maintain the robots spread along the curve, the proximity to the curve threshold and the curve radius, was an ongoing study.

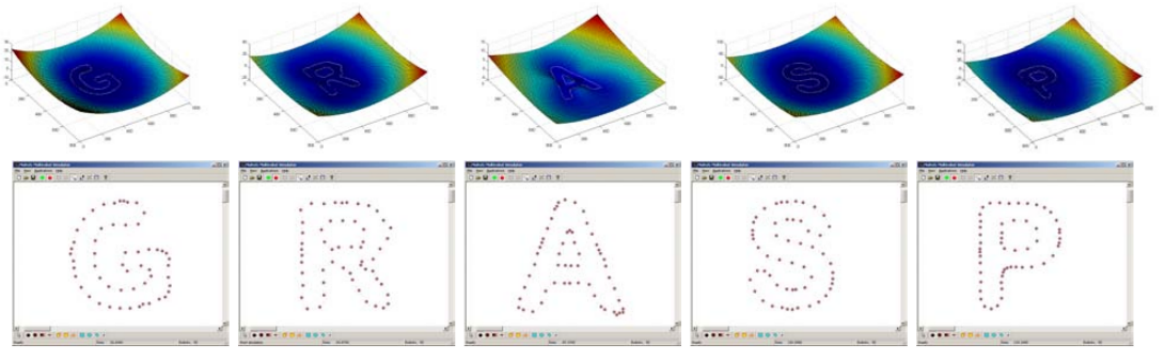
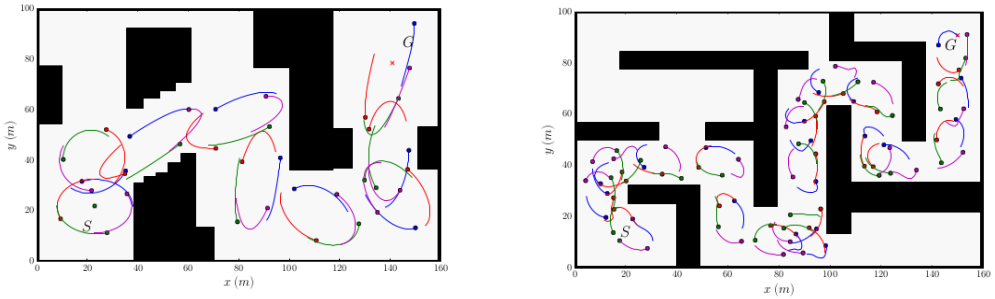


Figure 2.9: Simulations using 55 robots tracking different functions to form the letters G,R,A,S,P. On top, the isocontour of the functions generated for each letter, and on the bottom of each isocontour the robots - small red circles - spread along the shapes defined by the zero isocontour of the functions [Chaimowicz et al., 2005].

Another related problem approached in Saldaña et al. [2016] was the situation in which a coordinated team of robots must perform dynamic perimeter surveillance along the boundary of a deformable region Ω. The authors consider an environment $\varepsilon \subset \mathbb{R}^2$, which contains a set of static obstacles $\mathcal{O} \subset \mathbb{R}$. When it comes to trajectory following, two problems are considered on their work: coordinate the robots to track a defined, yet deformable, trajectory that defines Ω (the dynamic perimeter), and create a trajectory so that the previously defined region Ω is allowed to move, thus it is possible to monitor the entire region ε by Ω. The authors then define a parametric shape $\Phi : [0, 1] \times \mathcal{Q} \rightarrow \varepsilon$, where $\mathcal{Q}$ is the n-dimensional configuration space of the shape. Φ is a function that maps the connected set of the perimeter (denoted by $\partial\Omega$) into ε and can transform its shape based on n parameters as degrees of freedom. This deformation and how it can be used in order to perform navigation is presented in Figure 2.10. The problem

of creating a trajectory so the dynamic perimeter Ω is capable of moving from a start position to a goal position in ε avoiding the obstacles by deforming Ω's shape is solved by extending the standard kinematic dynamics motion planning technique proposed in LaValle and James J. Kuffner [2001].

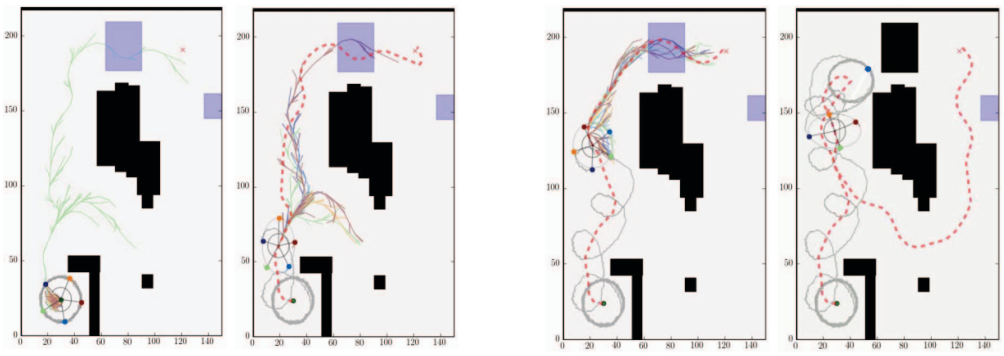


(a) Ellipse Shape (b) Crooked Egg Shape

Figure 2.10: Escort of (a) an ellipse defined shape with constant area, and (b) a crooked egg shape, from S to G in an environment with static obstacles represented by the black rectangles. The small circles represent the robots, and the continuous lines are their movement before the snapshot [Saldaña et al., 2016].

Jahn et al. [2017] proposes a distributed technique that allows a coordinated team of robots to plan the deformation of the boundary shape in order to escort the region within the designed boundary. Different from Saldaña et al. [2016], in which the authors coordinate the robots to track a defined, yet deformable, trajectory that defines Ω (the dynamic perimeter), [Jahn et al., 2017] deals with the problem of planning the perimeter evolution and the problem of controlling the motion of each of the individual robots involved in the surveillance task. It allows the robots to create a virtual fence, which aims to avoid internal or external agents crossing through the delimited area. Another contribution granted by the authors is the possibility of dealing with partially known or unknown environments, using only the information locally sensed by the robots in order to perform the surveillance task. To perform the surveillance task, each robot computes the trajectory of the shape between the start and the endpoint in the environment. Then, the best plan gets accepted as the one to be followed in an auction-like, decentralized procedure that consists of a token-ring communication topology where the token owner receives the bids for the auction during the current interval and broadcasts the winner which broadcasts' its solution to the team. Safety actions, like maximum braking in case a previously unknown obstacle is found on the trajectory, are applied. It is important to note that this kind of safety action is taken by all the robots in the team, with the intention to maintain the boundary function within the free configuration space and continuously differentiable so convergence can

be guaranteed. During the navigation, the robots continue to explore the configuration space trying to find improved trajectories, whilst they are controlled to circulate the perimeter of the designed shape in such a way that they keep an equal distance in the angle while only communication with their immediate neighbors considering the adopted ring topology. According to the authors, the main limitations are related to the consideration of a specific class of shapes that can be synthesized by the robots (like ellipses or more generally star-shapes) and the consideration of static obstacles. Figure 2.11 presents an example of how this approach can be used to guide a team of robots that navigate in a deformable formation through an unknown environment.



(a) (b)

Figure 2.11: Black rectangles represent known obstacles, and blue obstacles represent unknown obstacles. The robots are represented as colored circles. The committed trajectory is depicted as a red dashed line and each robot improved path is represented by the robot color. On (a), the original shape starts as a circle and then transforms itself into an ellipse to avoid the obstacle. On (b) the team discovers a new obstacle and re-plans its previously committed trajectory [Jahn et al., 2017].

2.3 Multi-Agent Task Assignment

When considering multiple agent systems, another common problem to be faced is how to allocate tasks to each agent in order to maximize the number of tasks and efficiently perform the ones that were assigned. In other words, given a set of tasks $\mathcal{T}$, and a set of agents $\mathcal{A}$, how to optimally assign a random task $\mathcal{T}_i$ to a random agent $\mathcal{A}_j$, so considering the characteristics of the agent and the properties of the task, there is no better assignment than $\mathcal{T}_i$ to $\mathcal{A}_j$? In those situations, it is obvious that the minimum amount of information necessary are the characteristics of the agent and the properties of the task in order to try to allocate tasks to agents in an optimal way.

2.3.1 Evolutionary Approach

On Jevtic et al. [2012], the task assignment considered consists of using a swarm of robots in a 2D arena to find specific targets. Using broadcast communication between them, the environment contains a set of tasks with the same or different importance, and robots equally capable of performing any task, but with the restriction of being able to perform exactly one task at a given time. More specifically, the tasks are targets in the arena with their associated qualities, which is defined by a scalar value that may represent the target importance or complexity, where higher values mean that more robots are necessary to perform the task. Three assumptions are necessary to describe the scenario and analyze the performance of the system due to the unpredictability of the robots' distribution before target allocation:

1. All targets are available for all robots. This is done by setting the broadcast communication range of the robots to cover the entire arena.

2. Once all targets in the arena are found, then the tasks are assigned, unless a specific group of robots is the only one to find a specific target, and in this case, the target's task is automatically allocated to the group. The total number of targets is preset and it depends on the experimental setup.

3. Reallocation to another target is not allowed.

By defining the utility of a robot as the probability of a robot k be allocated to a target t (p_t^k), the authors use a wheel-selection rule to allocate tasks, were the probability that the robot k is allocated to the target t is calculated by:

$$p_t^k = \frac{q_t^\alpha \eta_t^\beta}{\sum_{j=1}^{M} q_j^\alpha \eta_j^\beta}, \tag{2.3}$$

in which M is the number of targets,η is the utility value, q is the normalized value of the quality and both α and β are control parameters that allow the authors to bias the decision-making mechanism. As the authors state, their approach is based on a genetic algorithm, so one can consider the task allocation as the chromosome selection part of a generic genetic algorithm.

2.3.2 Auction Based Approach

Another common approach to task assignment problems are related to an auction-based approach. On this kind of methodology, tasks are considered items to be sold on

a market and agents are responsible for placing bids on an item (or a set of items). By standard rules of an auction, the owner of an auction is the one who places a higher bid on an item, the item cannot be divided and only one winner per item is allowed.

Gerkey and Mataric [2002] proposed the first online assignment architecture - named MURDOCH - which uses a first-price auction to assign each task. Their architecture consists of five steps:

1. **Task Announcement**: one agent acts as an auctioneer and publishes an announcement message for a task. This message contains information about the task and is addressed to the set of agents currently capable of performing the task based on the resources necessary to execute it.

2. **Metric Evaluation**: Since more than one agent can be able to perform a task, it is necessary a metric to determine which agent will be the one responsible to do it. In order to do it, the announcement also includes metric(s) with which each candidate robot can determine its fitness.

3. **Bid Submission**: After evaluating the appropriate metrics provide by the announcement, each agent places its bid based on their task-specific fitness score as a message.

4. **Close of Auction**: The authors determine that each auction round has a duration time. After the determined time, the auctioneer processes each received bid determines the winner, and notifies the bidders with a close message, notifying the end of that auction round. The winner is awarded a time-limited contract to execute the task while the losers return to listening for new tasks.

5. **Progress monitoring/contract renewal**: While the winner executes its awarded task, the auctioneer monitors its progress. Assuming that enough progress is being made, the auctioneer sends a message to the assigned agent to renew the task until it is completed. The assigned agent responds to the auctioneer's message with an acknowledgment message.

Based on those five steps, Gerkey and Mataric [2002] creates a centralized auction-based approach for task allocation. It is easy to understand why it is a centralized approach, given that there is one agent acting as an auctioneer responsible for managing tasks and a group of agents acting as bidders, and responsible for gathering the tasks and execute them.

On Michael et al. [2008], the authors propose a market-based coordination algorithm assuming that agents have knowledge of all tasks and the maximum number of

agents that can be assigned to every individual task. Also, all the agents are considered identical, the number of tasks is, at most, equal to the number of agents and a group of agents can be assigned to the same task, but one agent is capable of performing only one task at a time. On their work, the absence of a centralized auctioneer turns their market-based solution into a distributed market, where agents are able to bid for any task that they wish to be assigned. By using local communication among neighbors it is possible to assign tasks evaluating their availability and the bids placed by each agent. A task is considered available if it is not assigned to any agent. An example of how this market-based coordination works is depicted in Figure 2.12.

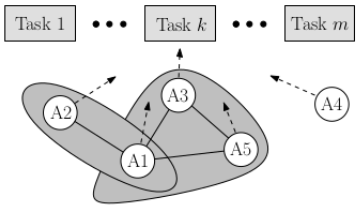


Figure 2.12: Agents A1 through A5, all requesting to be assigned to Task k. Solid lines indicate communication between the nodes. In this specific scenario, two local auctions involving two sets of agents $S_1 = \{A1, A2\}$ and $S_2 = \{A1, A3, A5\}$ are formed. Note that $S_1 \cap S_2 = A1$, so if $bid(A1) > max\{bid(A2), bid(A3), bid(A5)\}$, then A2, A3 and A5 will switch to other Task, and A1 and A4 will keep Task k. However, if $bid(A2) > bid(A1) > max\{bid(A3), bid(A5)\}$, then A1, A3 and A5 will switch to a different task [Michael et al., 2008].

2.4 Contextualization

Table 2.1 presents on each of its columns some characteristics of the problem that are addressed in this thesis and some of the related works mentioned in this chapter that were used as a reference.

When it comes to the characteristic of Task Assignment, the works of [Michael et al., 2008] and [Jevtic et al., 2012] are related to this work as our Task Assignment strategy can be seen as a combination of both techniques: we do approach the problem as an Allocation Problem – like it is done in [Michael et al., 2008] – at the same time that we use an algorithm to determine utility values and allocate the vehicles based on the value calculated by this algorithm – like it is done in [Jevtic et al., 2012].

On the context of Dynamic curves, whilst the works of [Chaimowicz et al., 2005] and Goncalves et al. [2010] relies on the use of radial basis and implicit functions, respectively, to deal with the problem of representing a curve, our work adopts a

methodology based on *Basis Splines* to represent the desired curve. Although the curve representation is done in different ways, it is important to mention that in all three cases dealing with a dynamic environment in which the curve representation changes and the vehicles navigating to or on it should adjust their route in order to go to or track the trajectory is a similar characteristic.

Finally, considering the works of Saldaña et al. [2016] and Jahn et al. [2017], one important characteristic that is shared between those works and this one is the ability of performing trajectory tracking. Given three different types of curve representation – Saldaña et al. [2016] defines a pre-computed yet deformable representation of a perimeter that must be tracked from the initial until the goal position, whilst Jahn et al. [2017] keeps the individual control of each of the robots so they can reach a consensus and define the shape and the deformation of the perimeter that will be guided from an initial to a goal position, and in our work we adopt a *Basis Spline* representation with the samples that define the knot vector with an undefined behavior – all three works present different techniques based on control theory to perform and consistently track a perimeter.

As a matter of fact, this work tends to unite part of the strategies adopted in the related works in order to create a new one that will have all the mentioned characteristics. Despite the fact that many of those characteristics were addressed in a very particular way as will be explained on Chapter 3, the key idea is to use part of their methodology and try to incorporate their ideas in this works' proposition to produce a new strategy which will have all the desired characteristics.

Table 2.1: Characteristics of the problem presented in this work and in related works.

	Multiple Agents	Multiple Curves	Task Assignment	Dynamic Curves	Trajectory Tracking
Chaimowicz et al. [2005]	X	-	-	X	-
Michael et al. [2008]	X	X	X	-	-
Goncalves et al. [2010]	-	-	-	X	X
Jevtic et al. [2012]	X	X	X	-	-
Saldaña et al. [2016]	X	X	-	X	X
Jahn et al. [2017]	X	X	-	X	X
This work	**X**	**X**	**X**	**X**	**X**

Chapter 3

Methodology

This chapter is dedicated to describing the methodology adopted to perform perimeter representation, task allocation and navigation tasks in scenarios composed by one or multiple robots and one or multiple perimeters.

As depicted in Figure 1.2, three parts of the pipeline – represented by rectangles – yield the problems that are addressed in this work. Those problems are usually associated with some inputs were they be an independent instance of objects or the solution of a previous problem. As an example, the first problem yield in the pipeline – related to the Perimeter Representation System – receives as input a set of clockwise-ordered points that do not require any kind of pre-processing, and provide a Perimeter as an output. On the other hand, the third problem on this pipeline receives as input the solution of the second problem. When a problem depends on the solution to another problem it means that it can't be solved without its previous solution. As an example, the third problem can't be solved without solving the second problem as the input of the third problem is the output of the second problem.

Assuming that all problems presented in this pipeline can be studied as a sub-problem, for a better presentation of each one of them they are separated as follow: Section 3.1 addresses the problem of representing a perimeter, Section 3.2 addresses the problem of allocating tasks to a group of robots, and Section 3.3 addresses how the navigation system is developed in order to perform the trajectory tracking task. Each section will formally present the problem, its restrictions and how they were developed in this work.

3.1 Perimeter Representation

The problem addressed in this section refers to the whole process of representing a perimeter given one or multiple sets of unique clockwise-ordered points. Those sets are hereby denoted as $\boldsymbol{\mathcal{P}_1}, \boldsymbol{\mathcal{P}_2}, \cdots, \boldsymbol{\mathcal{P}_m}$. It is important to notice that those perimeters are time-varying, thus each perimeter will be indexed by the set of points that define its boundary, and by a parameter of time $\mathbf{t}$, $\mathbf{t} \in \mathbb{N}^0$. A perimeter defined by the set of points $\boldsymbol{\mathcal{P}_k}$ in a given time $\mathbf{t}_i$ will be henceforth referred to as $\Omega_{\boldsymbol{\mathcal{P}_k}, \mathbf{t}_i}$.

3.1.1 Problem Statement

In this work, the Perimeter Representation Problem can be formally described as:

> Given multiple sets of points $(\boldsymbol{\mathcal{P}_1}, \boldsymbol{\mathcal{P}_2}, \cdots, \boldsymbol{\mathcal{P}_m})$ that represent time-varying coordinates of a $\mathcal{N}$-dimensional space in a given time $(\mathbf{t}_i)$, create a perimeter that has its boundary defined by the points of $\boldsymbol{\mathcal{P}_m}$ at time $\mathbf{t}$ $(\Omega_{\boldsymbol{\mathcal{P}_k}, \mathbf{t}_i})$.

In the multiple sets context, it is assumed that no perimeter is completely inside of another one. Formally, given multiple sets of points $\boldsymbol{\mathcal{P}_1}, \boldsymbol{\mathcal{P}_2}, \cdots, \boldsymbol{\mathcal{P}_m}$, if one set $\boldsymbol{\mathcal{P}_i}$ defines a closed perimeter, then **no** other perimeter $\boldsymbol{\mathcal{P}_k}$ $(\boldsymbol{\mathcal{P}_i} \neq \boldsymbol{\mathcal{P}_k})$ can be **totally** inside $\boldsymbol{\mathcal{P}_i}$. As an example, Figure 3.1 presents one instance of the problem in which three sets of points are used to determine three different perimeters.

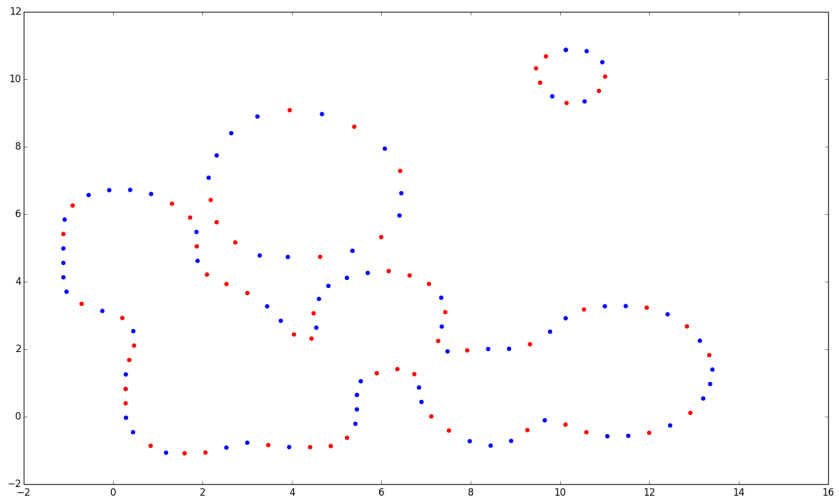


Figure 3.1: Problem instance with three different sets of points. Blue points are knot samples, received as input; and red points are calculated samples using De Boor's Algorithm.

3.1.2 Perimeter Definition

Before explaining the perimeter representation adopted in this work, it is important to describe a definition of what a perimeter is. One definition of what is a perimeter that fits the purpose of this work is to describe it as the continuous line forming the boundary of a closed geometric figure. Therefore, according to the previous definition, the sets depicted in Figure 3.1 does not represent a perimeter, as it is not a continuous line nor defines a closed geometric figure.

In order to properly represent a perimeter given a set of points at an arbitrary time (e.g.:, $\Omega_{\boldsymbol{\mathcal{P}_k},\mathbf{t}_i}$), it is necessary to, using the points contained in the specified set, create a closed geometric figure. To produce a well-delineated perimeter, this work relies on interpolating $\Omega_{\boldsymbol{\mathcal{P}_k},\mathbf{t}_i}$. As many interpolation methods can be used to perform such task, like polynomial interpolation, linear interpolation, piecewise constant interpolation, and spline interpolation, it is important that the selected method for this task provides a function that can guarantee that the resulting boundary presents properties such as smoothness, continuity and a small error when connecting the points of the set, in order to create perimeters that are easier for the agent to navigate in.

Also, it is extremely important that this interpolation procedure runs in a reduced computational time. Situations in which one or more perimeters must be recomputed should not take long in order to maintain the navigation in that region happening as unaffected as possible. This basically means that if an agent navigating through a region that is varying over time must perform a full stop on all its activities to recalculate its path, then this approach presents inefficiency regarding its planning phase. Also, in a multi-agent scenario, if an agent gets stuck deliberating, other agents will see him as an obstacle that must be avoided, hence compromising other agents' navigation task.

Given the desired properties of the interpolation method to represent a perimeter, in this work the adopted interpolation method is the *Basis Spline* method, also known as *B-Splines*.

3.1.3 *Basis Spline*

By definition, a *basis spline* is a spline function ($f(x)$) with minimal support. Considering the scenario studied in this work, $f(x)$ is a topological space - a set of points, along with a set of neighborhoods for each point - therefore the minimal support of $f(x)$ is the smallest closed set that contains all points that are not mapped to zero in $f(x)$. Formally, given a set of points X:

$$supp(f(x)) = \{x \in X | f(x) \neq 0\}. \tag{3.1}$$

According to De Boor [1986] a spline function of order n is a piecewise polynomial of order $n-1$. Being piecewise, the places where the pieces of this polynomial meet are known as **knots**. An important property regarding knots is the fact that they must always be sorted in non-decreasing order. A *basis spline* can be obtained by considering another representation of this spline, using a basis function of the same order as the spline function and defined over the same set of knots. Therefore all possible spline functions can be built as a linear combination of *basis splines*.

As explained, one of the most desired properties for the interpolation is it to be smooth in order to improve the quality of navigation. Also, according to Toraichi et al. [1987], the procedure for composing a spline function of order n that interpolates i input points can be done in a two-step procedure: computing the matrix A_i that transforms the spline coefficient vector into the sample value vector, and then calculating the values of the vector. Those procedures can be performed in computational complexity given by $\mathcal{O}(n^2)$ and $\mathcal{O}(n^2 \cdot i)$, respectively.

Other important characteristics granted by *basis spline* are related to the differentiability of the resulting polynomial pieces of each function. As explained in Gordon and Riesenfeld [1974], considering that each set $\boldsymbol{\mathcal{P}_k}$ contains unique points and they are used as the knot vector of the *basis spline*, the resulting *basis spline* of order n has its first $n-1$ derivatives of the polynomial pieces continuous across each knot. This is extremely useful as C^1 and C^2 continuity are granted, which means that the first and second derivatives of the equation are continuous. Another possible interpretation for C^1 and C^2 continuity is that the tangent vector - that measures speed - and acceleration vector are well behaved.

3.1.3.1 Using the *Basis Spline* representation

As *basis splines* were used to represent our perimeters due to its previously mentioned properties, it is important to describe how they were used.

Considering that each set $\boldsymbol{\mathcal{P}_k}$ contains unique points at time $\mathbf{t}_i$, and assuming that those points represent coordinates in $\mathbb{R}^2$, when a *basis spline* is used to interpolate the perimeter delimited by $\Omega_{\boldsymbol{\mathcal{P}_k},\mathbf{t}_i}$ considering $\boldsymbol{\mathcal{P}_k}$ as the knot vector, the resulting function will be henceforth named $\mathcal{B}(\Omega_{\boldsymbol{\mathcal{P}_k},\mathbf{t}_i})$. As the *basis spline* interpolation order n is a independent parameter, to guarantee C^2 continuity it is only necessary to choose $n \geq 2$. Therefore, as the computation of $\mathcal{B}(\Omega_{\boldsymbol{\mathcal{P}_k},\mathbf{t}_i})$ depend on the value of n - as explained in

Section 3.1.3 - it is important to choose a value for n that will not compromise the function computation.

Once $\mathcal{B}(\Omega_{\boldsymbol{\mathcal{P}_k},\mathbf{t}_i})$ is calculated, it is possible to compute N samples on $\mathcal{B}(\Omega_{\boldsymbol{\mathcal{P}_k},\mathbf{t}_i})$ using De Boor's Algorithm, which is, according to De Boor [1986], a numerically stable way for finding points on a *basis spline* curve. Like the knots used to create the *basis spline* interpolation, those samples are also $\mathbb{R}^2$ coordinates, and they are located in the perimeter boundary which is defined by the curve $\mathcal{B}(\Omega_{\boldsymbol{\mathcal{P}_k},\mathbf{t}_i})$. Therefore, those samples can be seen as coordinates that the robot must follow in order to patrol the region. It is also important to notice that since the knot vector is sorted - property granted by the *spline* - a set of samples between two knots is easy to sample considering the order from knot t_i to knot t_{i+1} and imposing a clockwise or counter-clockwise orientation.

Finally, having $\mathcal{B}(\Omega_{\boldsymbol{\mathcal{P}_k},\mathbf{t}_i})$ and all samples calculated, it is possible to start the robot movement. Considering the sampled points in a linear structure such as a vector, one approach to choose a goal point is to calculate the shortest distance between the robot's starting position and the samples. Once found, the sample that yields the shortest distance is considered the initial goal. Since samples are ordered due to the knot-order property of splines and considering the linear structure that they are stored, if the initial sample has index i, the following samples are $i+1, i+2, i+3, \ldots$. Also, considering n samples, it is possible to treat this structure in a circular way, being the $i = n+1$ position equivalent to the $i = 1$ position.

3.1.3.2 Using *Basis Spline* to represent multiple perimeters

It is straightforward that if multiple sets of points are given as input, each set should be treated separately and used to create its own *basis spline*. In this work, however, there are two separate scenarios that must be considered: a static perimeter and a dynamic perimeter scenario.

Considering static perimeters, once the samples are given as inputs, representing their region with a *basis spline* is just a matter of performing the interpolation. As those perimeters do not suffer any modifications in their samples positions, there is no need to perform another interpolation or any other operations regarding them, which means that once the *basis spline* representation is done, the first problem in the pipeline presented in Figure 1.2 is solved and does not need to be addressed anymore.

However, in the context of dynamic perimeters, two other situations must be taken into account: what to do when samples of two or more perimeters are too close to each other, and what to do if samples start to drift away from each other. Those situations are addressed in this work as *perimeter merge* and *perimeter split*

respectively.

3.1.3.3 Perimeter Merge

A *perimeter merge* is a situation that is probable to occur due to the fact that since perimeters dynamically change with an unknown behavior, there is a possibility of two or more perimeters intersect each other or to be so close to each other that there would be no safe path for the robot to navigate. In this work, all perimeter merges are handled by creating a new perimeter that will have on its boundaries points from the original perimeters. Also, points that lie inside the new perimeter are discarded as they are not useful nor should be visited since the goal is to navigate on the perimeter boundary.

In this work, a *perimeter merge* occurs when samples from different perimeters are within a threshold distance from each other. This threshold distance is defined considering the robot's dimensions: a circle capable of circumscribing the robot is described and if it touches more than one perimeter it means that those perimeters must be merged as the passage is narrow and if the robot tries to navigate through this passage it can be dangerous. Figure 3.2 illustrates two perimeters, $\Omega_{\boldsymbol{\mathcal{P}_1},\mathbf{t}_i}$ and $\Omega_{\boldsymbol{\mathcal{P}_2},\mathbf{t}_i}$, represented in black dashed lines, that should be merged:

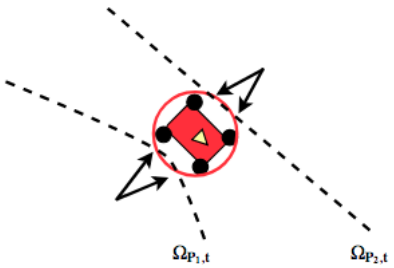


Figure 3.2: Two perimeters, $\Omega_{\boldsymbol{\mathcal{P}_1},\mathbf{t}_i}$ and $\Omega_{\boldsymbol{\mathcal{P}_2},\mathbf{t}_i}$, that must be merged given the dimensions of the robot.

When it comes to how to merge those two perimeters, many solutions are possible. At first, the most intuitive approach would be to create a Convex Hull giving as input all points. Although this approach would definitely create a boundary considering only the external points of the new perimeter, the result might be not so useful. By using a Convex Hull algorithm to determine the new points of the boundary, properties related to the concavity of the involved perimeters are neglected. Therefore, the result would produce a new set of points that provides an ineffective representation of the perimeter when interpolated.

Another possible solution is to create an alpha shape (α-shape) using all the points of the merge. A α-shape can be vaguely interpreted as the resulting shape formed by a set of points $S \in \mathbb{R}^n$. According to Edelsbrunner and Mücke [1992], α-shapes are a generalization of the Convex Hull of a point set. Given a set of points $S \in \mathbb{R}^n$ and a real number $0 \leq \alpha \leq \infty$, the α-shape of S is a geometric object with flat sides that is neither necessarily convex nor necessarily connected.

A more intuitive meaning of what is a α-shape, also explained in Edelsbrunner and Mücke [1992], is to compare the space $\mathbb{R}^n$ to a huge mass of ice-cream containing multiple hard chocolate sprinkles (the points in S). Using a spherical ice-cream spoon, the objective is to carve out all parts of ice-cream without touching the chocolate sprinkles, thereby even carving out ice-cream on the inside of the border. This process will eventually end up with an object bounded by caps, arcs, and points that is not necessarily convex. By straightening the round faces into triangles and line segments, the α-shape of S is obtained. In this case, α represents the radius of the spherical spoon. As $\alpha \to 0$ it is possible to see a degenerated solution that is the original set S. Otherwise, as $\alpha \to \infty$, it becomes more and more difficult to carve ice-cream between two chocolate sprinkles, and the α-shape becomes the Convex Hull.

Even though the α-shape seems like a promising technique for the perimeter merge, it also provides an ineffective solution. One of the reasons is the fact α-shapes will consider points that define a perimeter inside the outer-most perimeter as boundary points. As an example, Figure 3.3 shows the result of applying a α-shape technique over a set of points that resembles the letter "A":

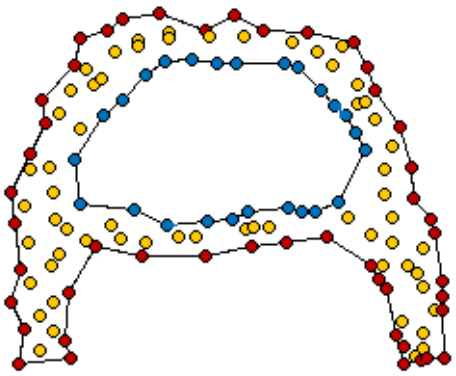


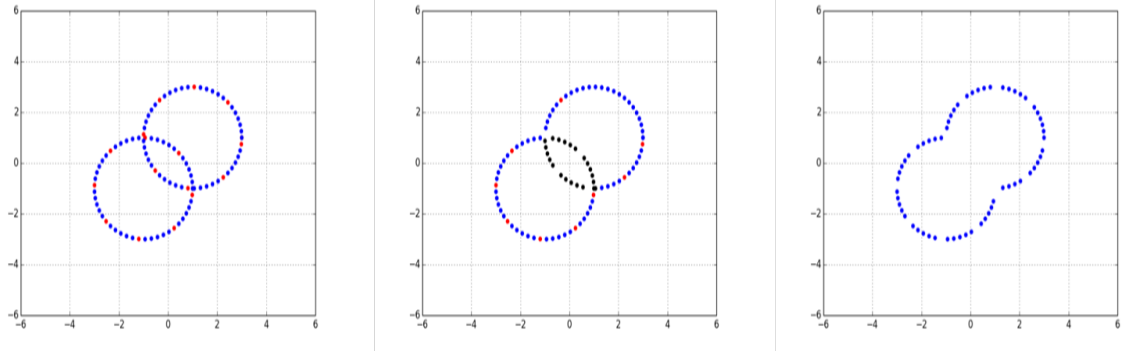
Figure 3.3: α-shape technique result over a set of points that resembles the letter "A".

The result displayed in Figure 3.3 shows that the points colored in red and blue are considered boundaries of the new perimeter, whilst only the yellow samples will be discarded. By assuming that two new perimeters will be created, it is possible to see

that the perimeter that uses the blue samples is totally inside the perimeter that uses the red samples. This violates the definition that no perimeter can be totally inside others. Another problem with the solution provided by the α-shape method is the fact that this technique is parameterized on the value of α. Therefore, it is not possible to choose a value for α that will definitely work for every possible merge situation. At this point, using the α-shape algorithm without any modification does not fulfills our necessities, but the fact that it can be used to generate different sets of boundaries being one of them the boundary that contains only the exterior-most points is useful for our purposes, as will be explained on the description of our algorithm for merge.

In this work, the merge algorithm is based on a modified version of the Convex Hull and the α-shape algorithms. This approach relies on computing the Convex Hull of the polygons involved and replace the edges used to join vertices from the same polygon in the Hull with the sequence of edges in the original polygons. Algorithm 1 shows the implemented technique, assuming that a generic Convex Hull algorithm is given.

The proposed merge algorithm take as inputs two sets of points, corresponding to two different perimeters. Its output is a set of points $\boldsymbol{\mathcal{P}_{i,j}}$ that contains the samples that define the boundary of a new perimeter, which is the merge of the intersecting ones. After the merging algorithm outputs the set $\boldsymbol{\mathcal{P}_{i,j}}$, a new perimeter $\Omega_{\boldsymbol{\mathcal{P}_{i,j}},\mathbf{t}_k}$ is defined, and the points in $\boldsymbol{\mathcal{P}_i}$ and $\boldsymbol{\mathcal{P}_j}$ that are not in $\boldsymbol{\mathcal{P}_{i,j}}$ are discarded. The next step is to perform a *basis spline* interpolation to create $\mathcal{B}(\Omega_{\boldsymbol{\mathcal{P}_{i,j}},\mathbf{t}_k})$. Figure 3.4 shows an example of the proposed algorithm's procedure, and Algorithm 1 shows its pseudo-code.



(a) Before Merge (b) During Merge (c) After Merge

Figure 3.4: Two regions before, during and after merging into one. Red points are calculated samples, blue points are knot samples, and black knots are the samples removed during the merge.

Algorithm 1: Merge Algorithm

Input: Two perimeters, $\Omega_{\boldsymbol{\mathcal{P}}_i,\mathbf{t}_k}$ and $\Omega_{\boldsymbol{\mathcal{P}}_j,\mathbf{t}_k}$
Result: A perimeter, $\Omega_{\boldsymbol{\mathcal{P}}_{i,j},\mathbf{t}_k}$, which is the merge of $\Omega_{\boldsymbol{\mathcal{P}}_i,\mathbf{t}_k}$ and $\Omega_{\boldsymbol{\mathcal{P}}_j,\mathbf{t}_k}$

```
1  allPoints ← Ω_{P_i,t_k}.points ∪ Ω_{P_j,t_k}.points
2  convex_hull ← ConvexHull(allPoints)
3  x, y, lastVertex, lastPol ← convex_hull[0]
4  Ω_{P_{i,j},t_k} ← [(x,y)]
5  i ← 1
6  while i < len(convex_hull) do
7      x, y, curVertex, curPol ← convex_hull[i]
8      polPoints ← list()
9      if curPol == lastPol then
10         if curPol == Ω_{P_i,t_k}.id then
11             vertexCounter ← len(Ω_{P_i,t_k}.points)
12             polPoints ← Ω_{P_i,t_k}.points
13         end
14         else
15             vertexCounter ← len(Ω_{P_j,t_k}.points)
16             polPoints ← Ω_{P_j,t_k}.points
17         end
18         vertex ← lastVertex
19         while vertex ! = curVertex do
20             vertex ← vertex + 1
21         end
22         if mod(vertex,2) ! = 0 then
23             Ω_{P_{i,j},t_k} ← Ω_{P_{i,j},t_k} + [(polPoints[vertex])]
24         end
25     end
26     else
27         Ω_{P_{i,j},t_k} ← Ω_{P_{i,j},t_k} + [(x,y)]
28     end
29     lastVertex ← curVertex
30     lastPol ← curPol
31     i ← i + 1
32 end
33 return Ω_{P_{i,j},t_k}
```

On the proposed merge algorithm, lines 9 until 23 are used to simulate the behavior of an α-shape algorithm, by selecting the vertexes that are not contained inside or in the intersection. The first part of that algorithm is to identify to which perimeter the vertex belong – the conditionals on lines 10 and 14. Once identified, the while loop on lines 19 to 22 is responsible for verifying if the points that compose that vertex are inside the polygon by checking how many times does that vertex intersects the

polygon. For an odd number, the samples can be considered outside the polygon, and added to the final set. For a pair number, it means that one of the samples is inside the polygon and therefore must be discarded. In order to select which sample to discard, it is possible to see that we iterate over all points on the previous calculated Convex-Hull (line 6), and treat each vertex individually.

The algorithm described before can also be described by analysing Figure 3.4. On the moment before the perimeter merge – Figure 3.4a – we have a set of samples that intersect each other. Those samples are distinguished between the sample from the original set – shown in blue – and the calculated samples – shown in red. For the matters of the merge algorithm, only the original set of samples is considered, therefore all red samples on the intersection or inside it are not considered. Once all samples that belong to the intersection or its interior are calculated – Figure 3.4b, the samples marked in black – they are removed and a new set containing only blue samples and that do not belong to the intersection or its interior will form a new perimeter, as shown on Figure 3.4c.

3.1.3.4 Perimeter Split

Another possible situation to happen due to the fact that perimeters dynamically change with an unknown behavior is a *perimeter split*. This situation can be seen as samples drifting apart from each other and their distance increasingly so significantly that between some of those samples there is enough space for the robot to pass. The space between samples that are used to check if there is enough room for the robot to pass is a parameterized value δ that can be adjusted only once considering the robot's dimensions and an additional safe distance. As an example, one can consider δ as the diameter of a circle that circumscribes the vehicle, similar to what is considered a value for perimeters to merge and explained in Section 3.1.3.3.

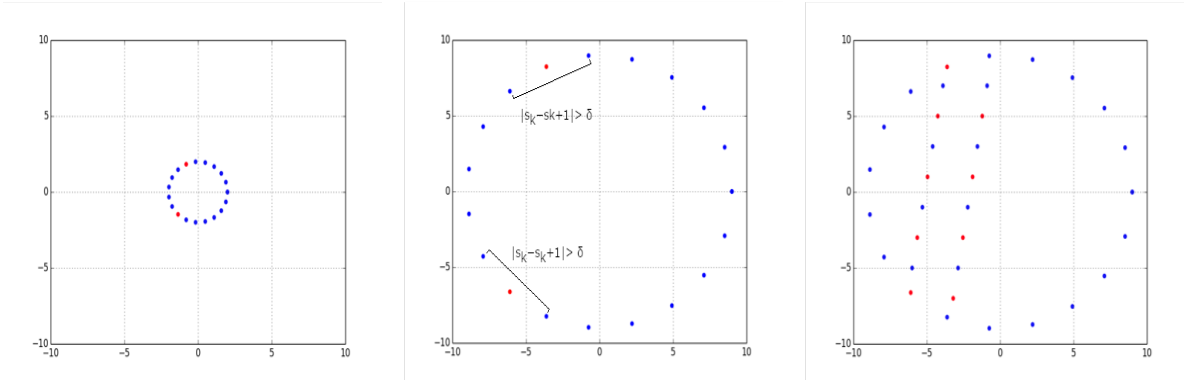
Although, having enough room for passing is not a sufficient condition to perform a split. It is important that if a perimeter splits into two others, both perimeters should be significant for the robot to cover. Therefore, to avoid degenerated cases such as just a small amount of samples separate and form a tiny new perimeter that does not cover a significant area, the split methodology adopted in this work considers that in order to split a perimeter the resulting areas delimited by the new sets of points of each perimeter must be significant. At this point, significant is considered an area that exceeds the robot's field of view, which means that the robot will have to navigate through the boundary in order to acquire information about it.

As an example, consider a perimeter $\Omega_{\mathbf{P_i},\mathbf{t}_k}$ in $\mathbb{R}^2$ and two of its samples (s_k, s_{k+1})

that are neighbors. As explained, consider that the distance between s_k and s_{k+1} is increasing until it is greater than a threshold value δ. Given that the distance between s_k and s_{k+1} is greater than a threshold value, i.e., $|s_k - s_{k+1}| > \delta$, the split process starts.

The split process consists in defining two sets, each containing one of the two samples, s_k or, s_{k+1}, and assigning the remaining samples of the perimeter to one of those sets based on their distance to s_k and s_{k+1}, where the shortest distance indicates to which set the sample belongs. If the two new sets define two perimeters with areas larger than the robot's field of view area, the split process is accepted and two new perimeters are created. If the area criteria are not met then the split process is interrupted and the perimeter remains as one. It is also important to notice that since the samples are the knot vector of the *basis spline* it is extremely rare to find cases in which a sample is equidistant from s_k and s_{k+1}. Though, and considering floating-point precision, if this case happens, the samples are assigned to the set that contains the less amount of samples, and in case both sets contain the same amount the assignment to one of the sets is random.

Algorithm 2 shows the pseudo-code of the split procedure adopted in this work. It receives as input a perimeter and it returns a list of perimeters containing the result of the split process. It is intuitive that if the split does not happen, then the return value is a list containing only the original perimeter without any modification. For better visualization of the code, the part which samples are assigned to one of the two split sets is described in Algorithm 3 and Figure 3.5 shows an example.



(a) Before Split (b) During Split (c) After Split

Figure 3.5: One perimeter that splits into two others. Red points are calculated samples and blue points are knot samples.

Algorithm 2: Split Algorithm

Input: A dictionary of perimeters, $\boldsymbol{\mathcal{L}}_{in}$, a threshold value δ for distance between points, a threshold value λ for the robot field of view size.

Result: A dictionary containing the result, $\boldsymbol{\mathcal{L}}_{out}$

$\boldsymbol{\mathcal{D}}_{out} \leftarrow$ dictionary()
$\boldsymbol{\mathcal{L}}_{out} \leftarrow \boldsymbol{\mathcal{L}}_{in}$
for $\Omega_{\mathbf{P_i},\mathbf{t}_k} \in \boldsymbol{\mathcal{L}}_{in}$ **do**
    i $\leftarrow 0$
    while i $<$ len($\Omega_{\mathbf{P_i},\mathbf{t}_k}$.points) **do**
        sample $\leftarrow \Omega_{\mathbf{P_i},\mathbf{t}_k}$.points[i]
        neighbour $\leftarrow \Omega_{\mathbf{P_i},\mathbf{t}_k}$.points[i+1]
        if ||sample - neighbour|| $< \delta$ **then**
            $\boldsymbol{\mathcal{D}}_{out}[\Omega_{\mathbf{P_i},\mathbf{t}_k}$.id] $\leftarrow$ (sample, neighbour)
            break
        end
    end
end
if $\boldsymbol{\mathcal{D}}_{out} \neq \emptyset$ **then**
    for key $\in \boldsymbol{\mathcal{D}}_{out}$.keys() **do**
        $\Omega_{\mathbf{fC},\mathbf{t}_k} \leftarrow$ list()
        $\Omega_{\mathbf{sC},\mathbf{t}_k} \leftarrow$ list()
        fS $\leftarrow \boldsymbol{\mathcal{D}}_{out}$[key][0]
        sS $\leftarrow \boldsymbol{\mathcal{D}}_{out}$[key][1]
        for sample $\in \boldsymbol{\mathcal{L}}_{in}$.[key].points **do**
            $\Omega_{\mathbf{fC},\mathbf{t}_k}, \Omega_{\mathbf{sC},\mathbf{t}_k} \leftarrow$ **assignSamples**($\Omega_{\mathbf{fC},\mathbf{t}_k}, \Omega_{\mathbf{sC},\mathbf{t}_k}$, fS, sS, sample)
        end
        firstCurve $\leftarrow \mathcal{B}(\Omega_{\mathbf{fC},\mathbf{t}_k})$
        secondCurve $\leftarrow \mathcal{B}(\Omega_{\mathbf{sC},\mathbf{t}_k})$
        if firstCurve.area() **and** secondCurve.area() $> \lambda$ **then**
            $\boldsymbol{\mathcal{L}}_{out} \leftarrow \boldsymbol{\mathcal{L}}_{out} \cup$ secondCurve $\cup$ firstCurve
            $\boldsymbol{\mathcal{L}}_{out} \leftarrow \boldsymbol{\mathcal{L}}_{out} - \boldsymbol{\mathcal{D}}_{out}$[key]
            for calculatedSample $\in$ firstCurve **do**
                if random.uniform(0, 1) > 0.5 **then**
                    firstCurve $\leftarrow$ calculatedSample
                end
            end
            for calculatedSample $\in$ secondCurve **do**
                if random.uniform(0, 1) > 0.5 **then**
                    secondCurve $\leftarrow$ calculatedSample
                end
            end
        end
    end
end
return $\boldsymbol{\mathcal{L}}_{out}$

Algorithm 3: Assign Samples

Input: Two lists of samples, $\Omega_{\mathbf{fC},\mathbf{t}_k}$ and $\Omega_{\mathbf{sC},\mathbf{t}_k}$, two samples, **fS** and **sS**, and **spl** a sample to be evaluated.

Result: Two lists of samples $\Omega_{\mathbf{fC},\mathbf{t}_k}$ and $\Omega_{\mathbf{sC},\mathbf{t}_k}$, in which one of those contains the new sample (**spl**).

```
1  if ||spl - fS|| < ||spl - sS|| then
2  |  Ω_fC,t_k ← Ω_fC,t_k ∪ spl
3  end
4  else if ||spl - fS|| > ||spl - sS|| then
5  |  Ω_sC,t_k ← Ω_sC,t_k ∪ spl
6  end
7  else
8  |  if len(Ω_fC,t_k) > len(Ω_sC,t_k) then
9  |  |  Ω_sC,t_k ← Ω_sC,t_k ∪ spl
10 |  end
11 |  else if len(Ω_fC,t_k) < len(Ω_sC,t_k) then
12 |  |  Ω_fC,t_k ← Ω_fC,t_k ∪ spl
13 |  end
14 |  else
15 |  |  if mod(random(),2) == 0 then
16 |  |  |  Ω_fC,t_k ← Ω_fC,t_k ∪ spl
17 |  |  end
18 |  |  else
19 |  |  |  Ω_sC,t_k ← Ω_sC,t_k ∪ spl
20 |  |  end
21 |  end
22 end
23 return Ω_fC,t_k, Ω_sC,t_k
```

Another way to understand how Algorithm 2 works is shown on Figure 3.5. On Figure 3.5a, we can see that the initial samples are all close enough, therefore that is no need to split the perimeter. On Figure 3.5b, two pairs of samples – which are indicated on the Figure – have their distance superior than the established threshold value ($|s_k - s_{k+1}| > \delta$), which indicates that a split should occur. To assign the samples to their respective group of the split, Algorithm 3 is used to determine to which group the samples belong based on (i): their distance to the evaluated samples (lines 1 to 6), (ii) the length of the resulting perimeters in case the first criteria gives the same output for both evaluated samples (lines 8 to 13), (iii) and finally a randomized decision in case both previous criteria results in the same output (lines 15 to 20). Finally, on Figure 3.5c, the split occurs and two perimeters are generated. As can be seem, the extremities of those perimeters are reconnected by use a *Basis Spline* interpolation. In order to

avoid this interpolation to occur again after the interpolation is performed given that the samples on the extremities can be distant from each other, some calculated samples are added to the knot vector, as shown on lines 28 to 37 of Algorithm 2.

3.1.4 Dynamic Behavior of a Perimeter

In this work, it is considered that all perimeters have a dynamic behavior. This dynamic behavior is characterized by a random movement on points of the set $\boldsymbol{\mathcal{P}_m}$, which makes their coordinates change. It is important to emphasize that those movements are random, which makes it impossible to create a model or to establish a pattern that would define the perimeter movement.

Considering those random movements on elements of $\boldsymbol{\mathcal{P}_m}$, the shape defined by the *basis spline* interpolation changes. Considering a set $\boldsymbol{\mathcal{P}_m}$ at time $\mathbf{t}_k$, it is important to guarantee properties such as C^2 continuity in the new function $\mathcal{B}(\Omega_{\boldsymbol{\mathcal{P}_m},\mathbf{t}_k})$. This property is guaranteed as the number of different knots in the knot vector will always be superior to 2 and the order n of a *basis spline* in this work is also always greater than two. Therefore, as explained in Section 3.1.3, the *basis spline* function is always at least twice differentiable and the C^2 continuity is granted.

It is also important to assume that, considering a maximum velocity for the robots the magnitude of the velocity at any point in the boundary is upper bounded by it. In the context of this work, this applies to all points in the knot-vector, as those points are the ones that are used in the interpolation for the definition of the boundary function. This assumption is necessary given the fact that robots have a limited velocity, and boundaries that change too fast would be impossible to track. It is even safer to assume that the robots move much faster than the boundary. This relationship is expressed in Equation 3.2, where $\|\dot{\mathrm{x}}(\mathbf{t})\|$ is the magnitude of the first derivative of the state vector of the robot in a given time $\mathbf{t}$:

$$\|\dot{\mathrm{x}}(\mathbf{t})\| \gg \left\| \frac{\partial \mathcal{B}(\Omega_{\mathbf{P}_m,\mathbf{t}})}{\partial \mathbf{t}} \right\| \tag{3.2}$$

3.2 Task Assignment

Assuming that all perimeters responsible for describing the regions in which the patrolling task will be performed are correctly created, the second problem yield in the pipeline described in Figure 1.2 is related to the allocation of patrolling tasks to each robot. This section addresses the problem of allocating those tasks, which can be seen as the strategy adopted to define which perimeter each robot will cover.

3.2.1 Problem Statement

In this work, a task is defined as the execution of a tracking mission on the boundary of a perimeter, which means that the robot must navigate to and circumnavigate a perimeter. Given this definition, in the context of this work the Task Assignment Problem can be defined as:

> Considering a set $\boldsymbol{\mathcal{B}_{all}}$ that contains all *basis splines* representing perimeters that should be under patrol, and a set $\boldsymbol{\mathcal{R}_{all}}$ that contains all robots available to perform the patrolling task, find an allocation of perimeters to robots, considering that one robot can only execute one perimeter patrolling at time, that maximizes a utility value $\mathcal{U}$. This utility value can express the ability of a robot to perform the desired task, or the cost to perform the task and many other properties.

In this work, a triple $(\mathcal{B}(\Omega_{\mathbf{P}_i,\mathbf{t}_j}), \mathcal{R}_k, \mathcal{U}_{\mathcal{B}(\Omega_{\mathbf{P}_i,\mathbf{t}_j}),\mathcal{R}_k})$ is the notation adopted to indicate that the patrolling task of a *basis spline* representation of a perimeter $\Omega_{\mathbf{P}_i,\mathbf{t}_j}$ is allocated to a robot $\mathcal{R}_k$ and the utility value of this allocation is given by $\mathcal{U}_{\mathcal{B}(\Omega_{\mathbf{P}_i,\mathbf{t}_j}),\mathcal{R}_k}$.

3.2.2 Task Assignment as an Allocation Problem

One solution to the Assignment Problem that is provided in this work is to model it as an Allocation Problem. Considering that the utility value $\mathcal{U}$ can be expressed as a cost for the robot to perform a given task, it is possible to define a reward that the robot will receive for accomplishing such task. This reward can be based on some characteristics of the task, such as the length of the perimeter, the total area to be covered, and others. After calculating a reward value for every perimeter, one solution is to allocate one or multiple robots to each perimeter considering the reward obtained and the cost to perform such task.

In this work, each robot is responsible for determining the visiting sequence of each perimeter. To determine it, a metric called *coverage efficiency* was created. The coverage efficiency is used to express how valuable it would be for a robot to cover a determined perimeter considering the time that the robot would take to arrive at its destination and the time that it would take to perform at least one circumnavigation over the perimeter.

An explanation on how to perform the calculation of the *coverage efficiency* metric is discussed in Section 3.2.3. Besides, it is important to notice that its calculation does not consider the dynamic behavior of the environment. Therefore, it is necessary to

establish how this dynamic behavior will affect the *coverage efficiency*. In this work, *coverage efficiency* is calculated before the robots start their mission, and recalculated for every perimeter merge or split that occurs. This decision is supported by the fact that the boundaries defined by the perimeters perform a bounded motion due to their velocity limitations, as explained in Section 3.1.4, and this property allow the assumption that any significant modification on the area defined by the perimeter occurs after a long period or after a split or merge.

After calculating the *coverage efficiency*, each perimeter is considered as a region to be tracked and the *coverage efficiency* is the utility value associated with it. Therefore, allocating an order of visitation for each perimeter to a robot based on this utility value is the allocation task proposed in this work.

3.2.3 Coverage Efficiency

As mentioned in Section 3.2.2, a metric, named *coverage efficiency*, is defined in this work to express how efficient it is for a robot to cover a determined perimeter. Considering a *basis spline* $\mathcal{B}(\Omega_{\boldsymbol{\mathcal{P}_k},\mathbf{t}_i})$ its *coverage efficiency* is given by Equation 3.3:

$$C_e = \frac{L}{E_{ct}}. \tag{3.3}$$

where L is the perimeter size and E_{ct} the *estimated coverage time*. In this work, the perimeter size is calculated based on the differences of the distance of all the samples that are part of the perimeter, not only the ones present in the knot-vector. Given that a *basis spline* is a piecewise polynomial function where the pieces of polynomial meet are its knots, and those knots are sorted into non-decreasing order, when samples are generated between two knots k_i and k_{i+1} those samples can be ordered in a clockwise order. Therefore, by defining one knot as the initial knot, all knots will be sorted due to this property, and consequently it is possible to calculate a perimeter as the distance between those samples. Hence, the perimeter size L given a total of m samples $(s_0, s_1, \ldots, s_m)$ on the *basis spline* is determined as:

$$L = \sum_{i=0}^{m-1} |s_i - s_{i+1}|. \tag{3.4}$$

The estimated coverage time (E_{ct}) is defined as the sum of the *coverage time* (ct) to transverse the perimeter and the *traveling time* (t_t) of the robot. The coverage time is estimated based on the perimeter size and the robot's maximum speed (v_{max}), as

shown on Equation (3.5):

$$ct = \frac{L}{v_{max}}. \tag{3.5}$$

The traveling time (t_t) is estimated considering the time necessary for the robot to navigate in a straight line at its maximum speed from its current pose until the entrance point of the target perimeter. It is important to mention that the traveling time does not take into account any obstacles or other curves on the robot's way to its target. Therefore, the traveling time (t_t) given the vector $\mathrm{x}(t_k)$ that represents the robots pose in at time t_k, the desired pose at the same time, $\mathrm{x}_d(t_k)$, and knowing v_{max}, can be calculated as:

$$t_t = \frac{\|\mathrm{x}(\mathbf{t}_k) - \mathrm{x}_d(\mathbf{t}_k)\|}{v_{max}}. \tag{3.6}$$

Finally, E_{ct} can be written as – considering N as the number of robots already allocated for that perimeter, and being at least 1 in case no vehicle are yet assigned to that perimeter:

$$E_{ct} = \frac{t_t + ct}{N}. \tag{3.7}$$

The *coverage efficiency* can also be described as a simple algorithm, that has as inputs a vehicle $\boldsymbol{\mathcal{R}}$ and a perimeter $\boldsymbol{\mathcal{D}}$, as shown on Algorithm 4.

Algorithm 4: Calculate Coverage Efficiency Algorithm

Input: A perimeter $\boldsymbol{\mathcal{D}}$, A vehicle $\boldsymbol{\mathcal{R}}$
Result: Coverage Efficiency Value (C_e) for $\boldsymbol{\mathcal{D}}$ given vehicle $\boldsymbol{\mathcal{R}}$

$L \leftarrow 0$
$i \leftarrow 0$
$s \leftarrow \boldsymbol{\mathcal{D}}$.samples()
$N \leftarrow \boldsymbol{\mathcal{D}}$.allocatedVehicles > 0 ? $\boldsymbol{\mathcal{D}}$.allocatedVehicles : 1
while $i <$ s.size() - 1 **do**
    $L \leftarrow L + |s_i - s_{i+1}|$
    $i \leftarrow i + 1$
end
$t_t \leftarrow \frac{\|\boldsymbol{\mathcal{R}}.\mathrm{x}(\mathbf{t}_k) - \boldsymbol{\mathcal{R}}.\mathrm{x}_d(\mathbf{t}_k)\|}{\boldsymbol{\mathcal{R}}.v_{max}}$
$ct \leftarrow \frac{L}{\boldsymbol{\mathcal{R}}.v_{max}}$
$E_{ct} \leftarrow \frac{t_t + ct}{N}$
$C_e \leftarrow \frac{L}{E_{ct}}$
return C_e

3.2.4 Task assignment based on the Coverage Efficiency

Considering each calculated coverage efficiency, as explained in Section 3.2.3, in this thesis we adopt a greedy strategy to define the visiting sequence of a perimeter by a robot. This strategy works by assigning to each robot the task to circumnavigate regions with a higher coverage efficiency ratio first.

After all vehicles calculate all the values of Coverage Efficiency for each of the respective perimeters that exists on the scenario on a given time, the allocation phase to define which vehicle will go to which perimeter starts.

On this phase, for each perimeter, we rank decreasingly the Coverage Efficiency value obtained by each vehicle for such perimeter, and we proceed with a greedy allocation strategy that first tries to allocate all perimeters – as shown on Algorithm 5 on lines 3 to 24. If all perimeters are allocated and we still have vehicles remaining, the reminiscent vehicles will recalculate their Coverage Efficiency metric – knowing that more than one vehicle can be allocated to a curve (thus, $N \geq 1$) – and a new allocation round, again based on a greedy strategy goes on – as shown on lines 25 to 36 of the same algorithm. It is important to notice that all ties are decided by a First Come, First Served fashion, which means that if two robots have the same value for Coverage Efficiency for a perimeter, the first one that requires to do the activity on that perimeter is the chosen one. It is also important to notice that Algorithm 4 is used on Algorithm 5 on lines 11 and 31 to calculate such value.

Algorithm 5: Algorithm for Assignment of a Perimeter Based on Coverage Efficiency

Input: A dictionary of perimeters, $\boldsymbol{\mathcal{D}}_{all}$, a list of all vehicles $\boldsymbol{\mathcal{R}}_{all}$ with their initial ($N = 1$) coverage efficiency values calculated for all perimeters in $\boldsymbol{\mathcal{D}}_{all}$

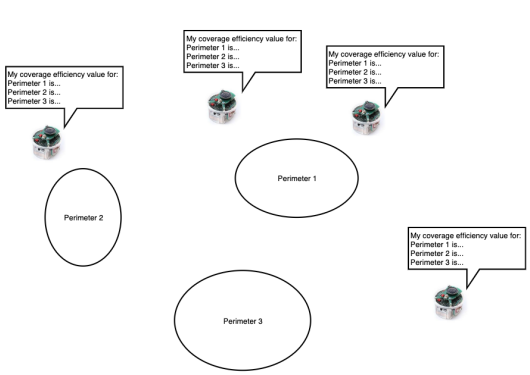
Result: An allocation for all vehicles in $\boldsymbol{\mathcal{R}}_{all}$

```
i ← 0
R_unassigned ← list()
while i < len(R_all) do
    if R_all[i].perimeter == NULL then
        R_all[i].covEffVal ← INT_MIN
        D_selected ← NULL
        for D in D_all do
            if CoverageEfficiency(R_all[i], D) > R_all[i].covEffVal then
                if D.isAssigned == false then
                    D_selected ← D
                    R_all[i].covEffVal ← CoverageEfficiency(R_all[i], D)
                end
            end
        end
        if D_selected != NULL then
            R_all[i].perimeter ← D_selected
            D_selected.isAssigned ← true
        end
        else
            R_unassigned ← R_all[i]
        end
    end
    i ← i + 1
end
for R in R_unassigned do
    R.covEffVal ← INT_MIN
    D_selected ← NULL
    for D in D_all do
        if CoverageEfficiency(R, D) > R.covEffVal then
            D_selected ← D
            R.covEffVal ← CoverageEfficiency(R, D)
        end
    end
    if D_selected != NULL then
        R.perimeter ← D_selected
    end
end
return R_all
```

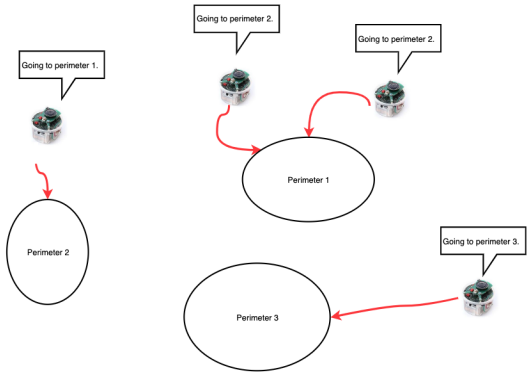
As the perimeter follows the restriction of bounded motion, explained in Section 3.1.4, its changes do not significantly affect the coverage efficiency metric. Due to this fact, the coverage efficiency is only recalculated in two situations: if a merge or a split occurs. In case a perimeter merge or split happens the coverage efficiency for all perimeters is recalculated. This recalculation can lead to three different scenarios on each robot:

1. Reorder the perimeter visiting sequence.
2. Assignment of a different task to a robot.
3. Assignment of the same task to the robot.

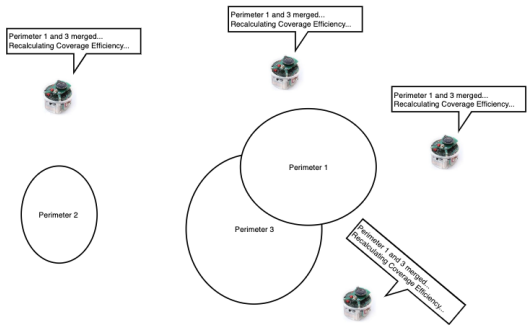
Figure 3.6 shows an example of how the task assignment based on the coverage efficiency works. At first, all robots calculate the coverage efficiency for all perimeters (a). Once this value is calculated, based on the highest value found, each robot is allocated to one of the perimeters (b), which can also lead to multiple robots being redirected to one perimeter. In this example, perimeters 1 and 3 merge, which leads to all robots recalculating their coverage efficiency value based on their current position and all the perimeters current state (c). Finally, all robots are reassigned to their respective perimeters according to their new value for coverage efficiency (d). It is also important to notice that the coverage efficiency takes into account how many vehicles are assigned to each perimeter



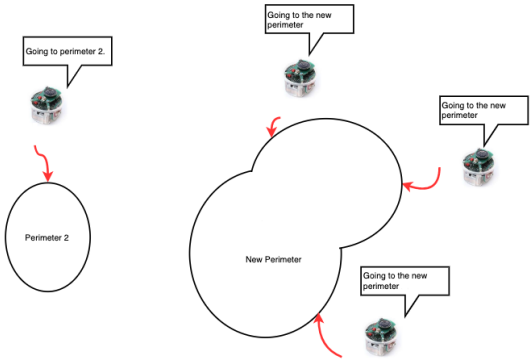
(a) Calculating coverage efficiency for the first time.



(b) Robots moving to their perimeters after calculating coverage efficiency.



(c) Merge between perimeters 1 and 3: all robots recalculate their coverage efficiency values.



(d) Robots are designated to their respective perimeters after merge and recalculating coverage efficiency.

Figure 3.6: Task assignment process based on coverage efficiency.

The perimeter visiting order changes when the length of a perimeter expands (shrinks) in a way that its coverage efficiency value increases (decreases) and becomes greater (less) than the values of coverage efficiency for other perimeters. This situation is likely to occur after a merge or split due to the fact that a new perimeter (or perimeters) will probably have its length value significantly different from its former value, and the number of perimeters after those operations are also likely to change.

The assignment of a different task to a robot occurs when, due to the value change of the coverage efficiency after a merge or split, one of the perimeters produces greater value for its coverage efficiency than the value of this metric for the current perimeter. In this case, and following the greedy strategy, the robot defines that its task is to follow the perimeter with the highest coverage efficiency, changing its goal.

Finally, an assignment of the same task is the situation in which after recalculating all the values for coverage efficiency, the greedy strategy results in the robot following its current perimeter. It is important to mention that if the perimeter is patrolled by

the robot is involved in the split or merge operation and yields the greatest value for the coverage efficiency, then the robot does not change its goal, and this is considered, in this work, as an assignment of the same task.

3.3 Navigation

Once a task is assigned to a robot or team of robots, the final problem of the pipeline described in Figure 1.2 must be solved. As explained, a task in this work is defined as one or multiple robots moving from their start position to its assigned path and performing a tracking mission by moving alongside the path. This task is completely related to the navigation system since the coordination of the robots for converging to their path and remain navigating alongside it, avoiding obstacles and collisions of any type.

3.3.1 Problem Statement

In this work, the problem yielded by the navigation phase can be formally defined as:

> Given a set of triples $\mathcal{N} = \langle \mathcal{B}(\Omega_{\mathbf{P}_i,\mathbf{t}_j}), \mathcal{R}_k, \mathcal{U}_{\mathcal{B}(\Omega_{\mathbf{P}_i,\mathbf{t}_j}),\mathcal{R}_k} \rangle$, in which $\mathcal{B}(\Omega_{\mathbf{P}_i,\mathbf{t}_j})$ is the *basis spline* function that corresponds to the perimeter to be patrolled, $\mathcal{R}_k$ is a robot or a team of robots assigned to perform the tracking task on $\mathcal{B}(\Omega_{\mathbf{P}_i,\mathbf{t}_j})$, and $\mathcal{U}_{\mathcal{B}(\Omega_{\mathbf{P}_i,\mathbf{t}_j}),\mathcal{R}_k}$ is the utility value that associates the gain of having a robot or a team of robots $\mathcal{R}_k$ performing the tracking task on the perimeter defined by the *basis spline* function $\mathcal{B}(\Omega_{\mathbf{P}_i,\mathbf{t}_j})$, establish a navigation strategy to guide each member of $\mathcal{R}_k$ to $\mathcal{B}(\Omega_{\mathbf{P}_i,\mathbf{t}_j})$, so the task can be performed.

3.3.2 Navigation Module Structure

The navigation system proposed in this work can be divided into three different modules: one responsible for collision avoidance, one responsible for planning the convergence to the desired path, and the one responsible for maintaining the vehicle tracking the assigned trajectory. Each of these modules is considered a control layer, and their priority is strictly top-down.

Assuming that the most desired characteristic for this navigation system is to provide secure navigation, the collision avoidance module plays the most important role in the system. Algorithm 6 shows an algorithm of how the navigation system modules work in order to avoid collision between two pair of vehicles :

Algorithm 6: Collision Avoidance Module

Input: A vehicle $\boldsymbol{\mathcal{R}_i}$,
A vehicle $\boldsymbol{\mathcal{R}_j}$,
A threshold value for collision distance δ (with $\delta > 1$),
A maximum speed value (MAX_SPEED) for the vehicles,
A minimum speed value (MIN_SPEED) for the vehicles
Result: Velocities $\boldsymbol{\mathcal{V}_i}$ and $\boldsymbol{\mathcal{V}_j}$

```
if distance(R_i, R_j) < δ then
    R_i.attractionForce ← distance(R_i.position, R_i.goalPosition)
    R_j.attractionForce ← distance(R_j.position, R_j.goalPosition)
    repelForce ← distance(R_i, R_j)
    R_i.resultantForce ← R_i.attractionForce + repelForce
    R_j.resultantForce ← R_j.attractionForce + repelForce
    if R_i.resultantForce < R_j.resultantForce then
        V_j.linVel ← MIN_SPEED
    end
    else
        V_i.linVel ← MIN_SPEED
    end
end
else
    V_i.linVel ← MAX_SPEED
    V_j.linVel ← MAX_SPEED
end
return V_i, V_j
```

3.3.3 Collision Avoidance Module

In this work, collision avoidance is implemented by using the Potential Fields technique. This technique is a widely known and used method for robot navigation, in which its main characteristics and limitations are explained in Koren and Borenstein [1991]. Regarding the use of this technique, in this work robots are modeled as charges with the same polarity, and each goal point in a given time $\mathbf{t}_k$ is a charge with different polarity. It is possible to see that, as robots have the same polarity, when two robots are close to each other they produce a repel force between them, whilst their force towards the goal is an attraction force. Also, in this work, the repel between a pair of robots is only significant if they are within a certain distance d from each other. This means that if two robots are farther than a distance d, their repel force is so small that it can be neglected.

Consider a resultant force $\boldsymbol{\mathcal{F}_{res}}$ given by the addition of the attraction force between one robot $\boldsymbol{\mathcal{R}_i}$ and its goal, $\boldsymbol{\mathcal{F}_A}$, and the repel force, $\boldsymbol{\mathcal{F}_R}$, produced by a pair

of robots ($\mathcal{R}_i$, $\mathcal{R}_k$). Figure 3.7 shows the force diagram on the red robot considering a close yellow robot and the curve samples represented by black dots.

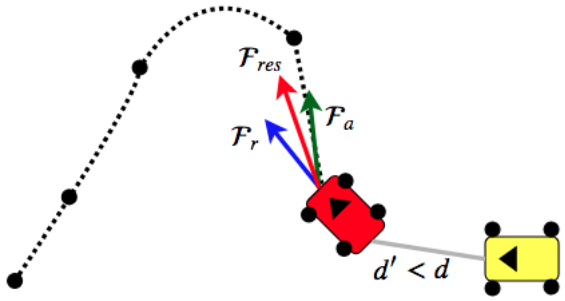


Figure 3.7: Force diagram for the red robot showing its vectors due to its interaction with the yellow robot. The green vector represents the attraction force towards the next point of the B-Spline, blue vector represents the repel force caused by the yellow robot, and the red vector is the resultant force.

As the proposed environment in this work is considered obstacle-free, which means that there are no objects that the robots must avoid, the only concern regarding collisions lies in the possibility of robots colliding with each other. Therefore, to avoid a collision, this module output desired angular and linear velocities for the robot so it can keep tracking the assigned perimeter and avoid collisions.

This work considers differential vehicles due to the fact that they are the most representative type of vehicles considering actual robots, and due to their applicability in perimeter tracking. The kinematic model adopted for simulations is presented on Equation (3.8):

$$\begin{bmatrix} \dot{x_c} \\ \dot{y_c} \\ \dot{\theta_c} \end{bmatrix} = \begin{bmatrix} \cos(\theta_c) & 0 \\ \sin(\theta_c) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_c \\ \omega_c \end{bmatrix}. \tag{3.8}$$

According to Equation (3.8), the coordinates (x_c, y_c) are the position reference of the vehicle on the 2D-space defined by the coordinates (x, y) and θ_c is the angle between the vehicle's velocity vector and the x axis. The control input is given by v_c, the linear velocity, and ω_c the angular velocity of the vehicle, hence the velocity commands can be expressed according to Equation (3.9)

$$\begin{bmatrix} \dot{x_c} \\ \dot{y_c} \\ \dot{\theta_c} \end{bmatrix} = \begin{bmatrix} v_c \cdot \cos(\theta_c) \\ v_c \cdot \sin(\theta_c) \\ \omega_c \end{bmatrix} \tag{3.9}$$

Finally, considering that the angular and linear velocities are inputs of the system and partially calculated by the Collision Avoidance Module, Section 3.3.4 addresses how those velocities are used by the robot so it can converge to and track a perimeter.

3.3.4 Trajectory Convergence and Tracking Module

One of the simplest ways to perform a trajectory tracking control design is based on tangent linearization along the reference trajectory. This type of controller relies on the idea of leading the state tracking error to zero. In this work, this controller is created considering Equation (3.10) as the state tracking error:

$$e = \begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_d - x \\ y_d - y \\ \theta_d - \theta \end{bmatrix} \tag{3.10}$$

where $[x_d, y_d, \theta_d]^T$ are the coordinates of the reference path and $[x, y, \theta]^T$ are the coordinates of the controlled robot. Considering the kinematic relations given in Equation (3.9), and by differentiating the error dynamics expressed in Equation (3.10), Equation (3.11) is obtained:

$$\dot{e} = \begin{bmatrix} \dot{e_x} \\ \dot{e_y} \\ \dot{e_\theta} \end{bmatrix} = \begin{bmatrix} \cos(e_\theta) & 0 \\ \sin(e_\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_d \\ \omega_d \end{bmatrix} + \begin{bmatrix} -1 & e_y \\ 0 & -e_x \\ 0 & -1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}. \tag{3.11}$$

Considering that the control inputs of the system are all based on the linear and angular velocities - as can be seen on Equation (3.9) - with respect to Equation 3.11, the input is defined according to Equation (3.12):

$$\begin{aligned} u_1 &= v_d \cdot \cos(e_\theta) - v, \\ u_2 &= \omega_d - \omega, \end{aligned} \tag{3.12}$$

where v_d and ω_d are feed-forward signals – addressed on Equations (3.23) and (3.24) – and v and ω are control loop system signals. Those signals can be rewritten according to Equation (3.13):

$$\begin{aligned} v &= v_d \cdot \cos(e_\theta) - u_1, \\ \omega &= \omega_d - u_2, \end{aligned} \tag{3.13}$$

and the state tracking error dynamics in Equation (3.11) can be rewritten according to Equation (3.14):

$$\dot{e} = \begin{bmatrix} \dot{e_x} \\ \dot{e_y} \\ \dot{e_\theta} \end{bmatrix} = \begin{bmatrix} 0 & u_2 & 0 \\ -u_2 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} + \begin{bmatrix} 0 \\ \sin e_\theta \\ 0 \end{bmatrix} \cdot v_d + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}. \tag{3.14}$$

In order to produce a controller for this system, first, we introduce its structure. Figure 3.8 presents the control loop structure used in this work:

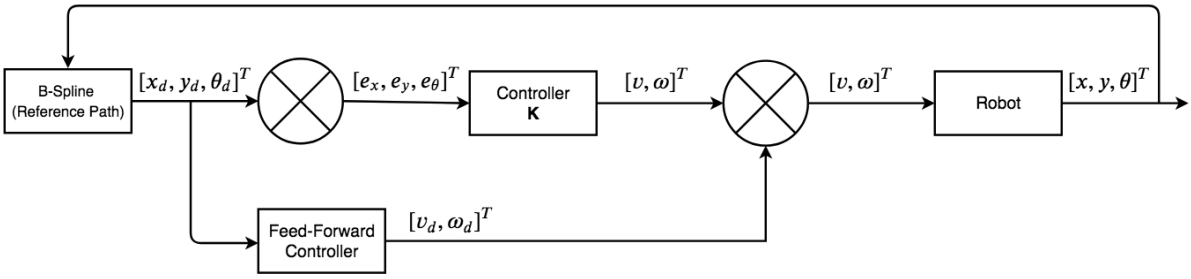


Figure 3.8: Control Loop Structure.

Since it is desired to control the position error of the system ($[e_x, e_y, e_\theta]^T$) using two control inputs ($[v, \omega]^T$), a proportional controller is used. The structure of this controller is defined based on the following ideas:

1. To decrease the error in x (e_x), the action should be on the linear velocity of the system.
2. To decrease the error in θ (e_θ), the action should be on the angular velocity of the system.
3. Due to the fact that system used is a differential robot, the error in y (e_y) can be decreased by acting in the angular velocity of the system.

Considering those three statements, the proportional controller **K** used in this work can be formulated according to Equation (3.15):

$$\begin{bmatrix} \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} -k_1 & 0 & 0 \\ 0 & k_2 & k_3 \end{bmatrix} \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix}. \tag{3.15}$$

In order to obtain the values for k_1, k_2 and k_3, we first propose a linearization of the system presented in in Equation (3.11). This linearization is only possible once

we assume that the vehicles non-linear dynamic's is too slow. Once assumed that, by linearizing the system at $e_x = e_y = e_\theta = 0$ and $v, \omega = 0$, Equation (3.16) is obtained:

$$\Delta\dot{e} = \begin{bmatrix} 0 & \omega_d & 0 \\ -\omega_d & 0 & v_d \\ 0 & 0 & 0 \end{bmatrix} \cdot \Delta\vec{e} + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \cdot \Delta v. \tag{3.16}$$

Now that we have a linearized system that describes our model, it is possible to apply Laplace's transformation and define the parameters k_1, k_2 and k_3, by using a third-order characteristic equation. A desired characteristic equation of a third-order system is given by Equation (3.17):

$$f(s) = (s + 2\xi\omega_n) \cdot (s^2 + 2\xi\omega_n s + \omega_n^2), \quad \xi, \omega_n > 0, \tag{3.17}$$

having three constant eigenvalues being one negative and real (at $s = -2\xi\omega_n$) and a complex pair with natural angular frequency (ω_n) strictly positive ($\omega_n > 0$) and a damping coefficient $\xi \in (0, 1)$.

Considering our closed loop system using the proposed controller $\mathbf{K}$, it is possible to calculate its characteristic equation by using the pole allocation technique. This can be achieved by solving the following equation:

$$g(s) = \det(s\mathrm{I} - \mathrm{A} + \mathrm{B} \cdot \mathrm{K}) \tag{3.18}$$

where I is the identity matrix, A, is the system matrix, B is the input matrix and K is the controller matrix. Solving Equation (3.18), the characteristic equation $g(s)$ is given by:

$$g(s) = s^3 + (k_1 + k_3) \cdot s^2 + (k_1 k_3 + k_2 v_d + \omega_d^2)s + k_1 k_2 v_d + k_3 \omega_d^2. \tag{3.19}$$

By comparing the polynomials obtained in Equations (3.17) and (3.19) it is possible to establish a relationship between the values of ξ and ω_n and the values of k_1, k_2 and k_3 in order to estimate the parameters of the controller, as shown on Equation (3.20).

$$\begin{aligned} 4\xi\omega_n &= k_1 + k_3, \\ 4\xi^2\omega_n^2 + \omega_n^2 &= k_1 k_3 + k_2 v_d + \omega_d^2, \\ 2\xi\omega_n^3 &= k_1 k_2 v_d + k_3 \omega_d. \end{aligned} \tag{3.20}$$

According to Equation (3.20), one possible configuration for the gains k_1 and k_2 is:

$$k_1 = k_3 = 2\xi\omega_n \tag{3.21}$$

and this choice yields for k_2 the following expression:

$$k_2 = \frac{\omega_n^2 - \omega_d^2}{|v_d|}. \tag{3.22}$$

It is important to notice that due to the conditions implied for ω_n and ξ in Equation (3.17), k_1, k_2 and k_3 are strictly positive constants. As the values for k_1, k_2 and k_3 must be positive, it can be seen that, for k_2, two conditions are necessary to satisfy the previous condition:

1. The natural frequency of the system must be higher than the max angular speed.
2. As v_d tends to 0, k_2 tends to infinity.

Assuming that the vehicle must follow the Cartesian trajectory defined by the *Basis Spline* set of points in a given time t – $(x_d(t), y_d(t))$, with $t \in [0, T]$ (possibly, $T \to \infty$) – from the kinematic model of the vehicle presented in Equatino (3.9), one can define nominal feed-forward commands for $v_d(t)$ and $w_d(t)$ as shown in Equations (3.23) and (3.24) respectively:

$$v_d(t) = \pm\sqrt{\dot{x}_d^2(t) + \dot{y}_d^2(t)}, \tag{3.23}$$

$$\omega_d(t) = \frac{\ddot{y}_d(t)\dot{x}_d(t) - \ddot{x}_d(t)\dot{y}_d(t)}{\dot{x}_d^2(t) + \dot{y}_d^2(t)}, \tag{3.24}$$

where the chosen sign for $v_d(t)$ will determine forward or backward motion of the vehicle. It is also important to notice that this is possible given the fact that our *Basis Spline* representation guarantees continuity in C^2, which allow us to differentiate it twice at any given time in $[0, T]$.

3.3.4.1 System Stability Analysis

As the parameters were decided, it is necessary to perform a stability analysis of the system. By using the parameters assigned by Equations (3.21) and (3.22), as velocity decreases the parameters k_1, k_2 and k_3 tends to zero. This makes the system not controllable. Therefore it is necessary to prove its stability. In this work, the proposed analysis of stability is based on Lyapunov Theorem and LaSalle's Invariance Principle and follows.

Consider the following Lyapunov candidate function proposed in Equation (3.25), which is based on the controller proposed in Equation (3.15) and on the idea of leading the state tracking error to zero:

$$V(e) = (e_x^2 + e_y^2) \cdot \frac{k_2}{2} + \frac{e_\theta^2}{2} \tag{3.25}$$

Then, considering the linearized model presented in Equation (3.16), and the controller proposed in Equation (3.15), the time derivative of the Lyapunov candidate function shown in Equation (3.25) is given by:

$$\dot{V}(e) = -k_1 k_2 e_x^2 - k_3 e_\theta^2. \tag{3.26}$$

According to Lyapunov's Theorem and using LaSalle's Invariance Principle, a valid Lyapunov function – $V(e)$ – to investigate stability of a system must be semi-positive definite, continuous, be continuous within its derivatives on all its components and be radially unbounded. Furthermore, it is necessary that the first derivative of the candidate function is negative semi-definite ($\dot{V}(e) \leq 0$). If all those conditions are satisfied, then the system can be said to be stable.

As can be seen on Equation (3.25), the positive semi-definite property is granted since parameter k_2 is strictly positive and all the remaining terms – e_x, e_y and e_θ – are real values and squared, thus positive. Also, the used candidate function presents a first derivative which is negative semi-definite. Since $k_2 \neq 0$, the equality $V(e) = 0$ holds for $V(0)$. Finally, it is possible to see that $V(e)$ is bounded and tend to some limit value. Using Barbalat lemma, $\dot{V}$ tends to zero. From this and analyzing the system equations, it is possbiel to show that $(v_d^2 + \omega_d^2) \cdot e_i^2$ (with i $= x, y, \theta$) tends to zero so that, from the persistency of the (state) trajectory, the result follows.

3.3.5 Navigation Strategy Based on Coverage Efficiency

Once it is shown that the system adopted in this work is stable, the strategy created to accomplish the trajectory tracking task based on the Coverage Efficiency of each perimeter – metric described in Section 3.3.5 to define in which order each perimeter should be visited – is explained in this section.

After calculating each coverage efficiency given the set of perimeters to cover, the proposed strategy proceeds by estimating – based on the current distribution of the samples in the environment – the closest pair of samples between two perimeters. Those points are henceforth named exit points. Each exit point belongs to one perimeter, and

a pair of exit points will never be made from points of the same perimeter.

Each pair of exit points contains the coordinates of samples in two distinct perimeters and are used to determine the position in which a robot should ideally leave one perimeter and enter the other in order to swap between perimeters and covering each one of them. As an example, consider the scenario presented in Figure 3.9, containing three different perimeters and their exit points:

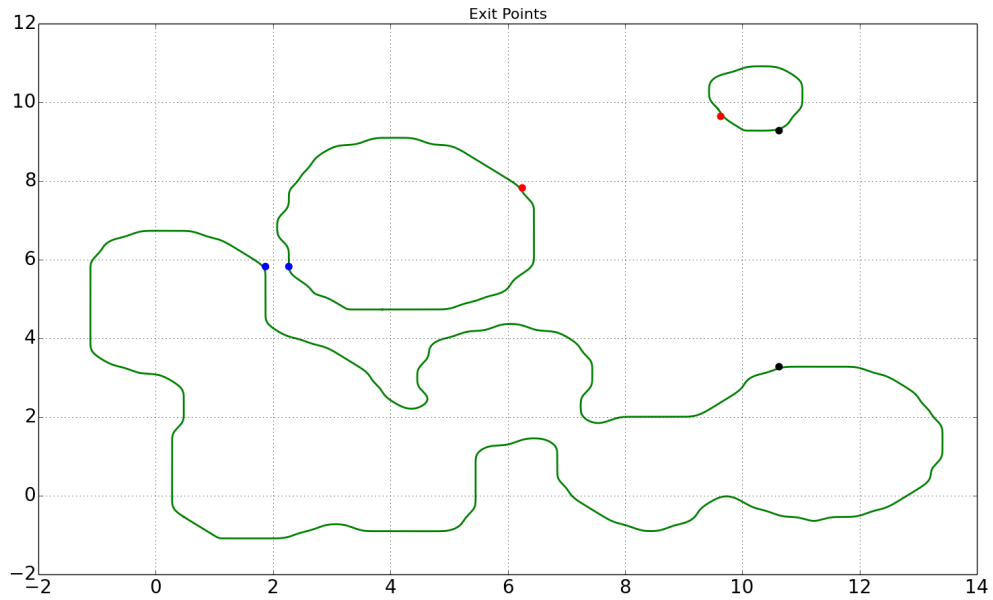


Figure 3.9: Three pairs of Exit Points in three different perimeters. Each pair is described in different colors (blue, red and black).

As shown in Figure 3.9, each pair of exit points are marked using a color – e.g.: the exit points between the two largest areas are marked in blue. Those pairs are calculated based on the euclidean distance of samples in a given time, meaning that the two closest samples between two perimeters will be selected as the exit points. It is important to notice that in the context of static perimeters, finding the exit points is only necessary to be performed once, as samples will not move and there is no need to recalculate the perimeters.

However, in the context of dynamic perimeters, when perimeter merges or splits occurs, it is necessary to recalculate the exit points like the ones that were previously calculated may have changed due to the necessity of merging or splitting two or more regions.

As the exit points are based on the samples of the *basis spline* and the coordinates of a sample tend to vary randomly in order to simulate the perturbation on the

environment, the solution proposed in this work is to determine the exit points based on samples by using the knot vector of the *basis spline*. In the context of dynamic perimeters instead of saving the coordinates of the exit points, it is more efficient to save their position on the knot vector of the basis spline. Therefore, as explained in Section 3.1.3.1, once performed the *basis spline* interpolation the knot vector will be defined, and the position of each knot on the vector will be fixed until a new interpolation occurs, which means that all perturbations on the system will maintain the knots in their position in the knot vector. Hence, it is not necessary to recalculate the position of an exit point when modifications happen on the curve as long as the position of the sample in the knot vector is known.

As the vehicle approaches the position of an exit point, it is important to define when change from one perimeter to another, and when to stay covering the perimeter that the vehicle is currently in.

In this case, the strategy adopted in this work is to apply the Coverage Efficiency algorithm proposed in Section 3.2.3, but instead of considering all m samples to calculate the perimeter size L, only the samples that were not visited are considered. One possible approach to calculate the Coverage Efficiency considering only the non-visited samples is to subtract the distance defined by the samples that were already visited by the robot from its perimeter size. Consider a set V that contains the coordinates of all visited samples by the robot until it reaches the exit point. For the *Coverage Efficiency* of this robot, Equation (3.4) can be rewritten as:

$$L = \sum_{i=0}^{m-1} |s_i - s_{i+1}| - \sum_{i=0}^{|V|-1} |v_i - v_{i+1}|. \tag{3.27}$$

The same idea used to verify exit points by their positions in the knot vector of a *basis spline* can be used in order to create a set of visited samples. Instead of saving the coordinates of the positions that were visited by the vehicle, it is preferable to save the position of the samples in the knot vector that were visited, due to the possibility of – in a dynamic perimeter scenario – keeping the position of the sample updated, thus the coverage efficiency can be calculated considering the current shape of the perimeter, and estimate the covered perimeter.

Chapter 4

Experiments and Results

This chapter is dedicated to present the results obtained by applying the methodology proposed in Chapter 3 for the perimeter representation and tracking task. For all experiments in this section, we analyze the behavior of the actuators of each vehicle, the linear velocity actuator and the angular velocity actuator, and the tracking error in both x and y axis. In order to provide a scenario in which the simulations reproduces the behavior of actual robots, we limited the value for the controllers to 0.5m/s for the linear velocity whilst the angular velocity is limited to $\frac{\Pi}{8}$rad/s. Also, we analyze the behavior of the interpolation by showing different degrees of interpolation for the same curve and what is the computational time cost for each of them.

Section 4.1 shows different sets of samples and the cost and result of interpolating each of them using *Basis Spline*. In this section we will address the problem of using the *Basis Spline* with degree 20 and a maximum number of 1000 calculated samples to generate our desired continuous path given the original set of samples. We also present an experiment with a reduced set of samples to compare the impact caused by the number of samples on the result of the interpolation.

On Section 4.2 we show the results obtained when using four robots to perform perimeter tracking on a dynamic perimeter considering a scenario in which the perimeter is expanding under severe perturbation, meaning that although the speed of the vehicle and the boundary obeys Equation 3.2, the boundary speed change ratio will be set to a significant value. An analogous experiment, in the same section, uses the opposite scenario: this time instead of expanding, the perimeter is shrinking.

Section 4.3 addresses the problem for four robots navigating on a perimeter that has an unknown behavior and splits into two different ones. Here we will explore the allocation and tracking of the vehicles once the perimeter split occurs.

On Section 4.4, we explore the opposite scenario shown on Section 4.3. This

time, two perimeters represented by circles will merge into a single perimeter as their boundaries intersect each other. In this experiment, we aim to explore the behavior of the four vehicles as their original perimeters merge into a single one.

In Section 4.5, two experiments are presented in order to verify what is the behavior of the vehicles when multiple static perimeters are part of the scenario. Those two experiments also aims to study what happens when a perimeter is on the way of a vehicle that must track a different perimeter.

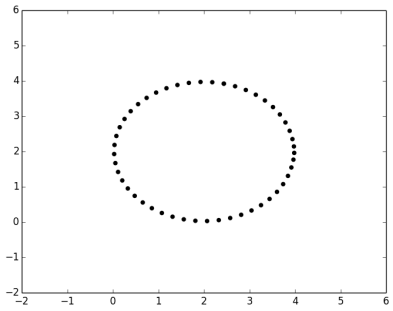
Section 4.6, we study the behavior of multiple dynamic perimeters in which multiple perimeters have their samples being under some disturbance. In the experiment proposed in this section, we aim to bring the whole picture of this work: a dynamic perimeter behavior for all perimeters, a merge situation, and a split situation.

Finally, in Section 4.7, we present an analysis of the *Coverage Efficiency* metric based on experiments of Sections 4.3, 4.5 and 4.6, in which we discuss how our proposed metric can be used as a tool for providing an strategy that favors covering as many areas as possible.

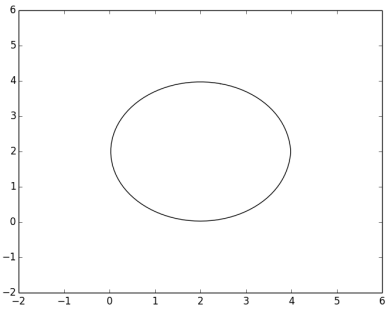
4.1 Perimeter Representation Using B-Splines

As mentioned on Chapter 3.1, in order to create paths given a set of points, this work adopted the *Basis Spline* interpolation method. As *B-Splines* provide smooth and well fit curves, and the computational cost to create them does not seem to compromise the overall performance, choosing it as the interpolation method to create paths that will represent the regions of interest is the solution adopted in this work.

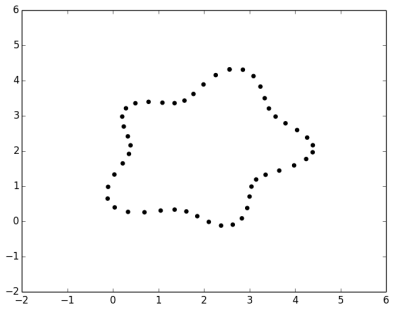
Once they were adopted, the primary concern in this work was regarding the polynomial degree that should be used to perform the interpolation. As defined on Chapter 3.1.3, the complexity of the interpolation is strongly related to the degree of the desired polynomial used for it. Therefore, in this experiment by using a polynomial degree of 20, we show the results achieved by the initial interpolation of the scattered samples in Figure 4.1 and Table 4.1 shows the result of the time needed to perform the interpolation given the desired polynomial degree for all initial regions explored in experiments 1 to 6 in this chapter.



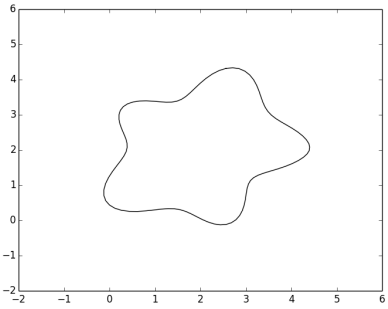
(a) Experiments 1 and 3 Initial Samples.



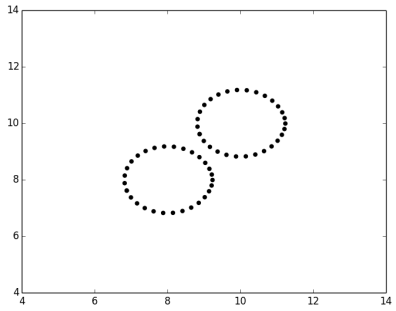
(b) Experiments 1 and 3 Interpolation.



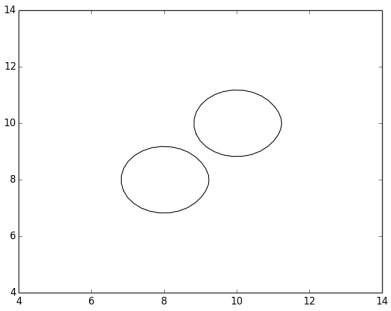
(c) Experiment 2 Initial Samples.



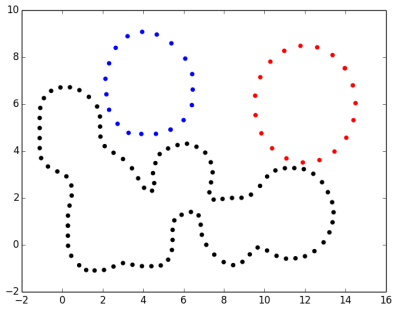
(d) Experiment 2 Initial Interpolation.



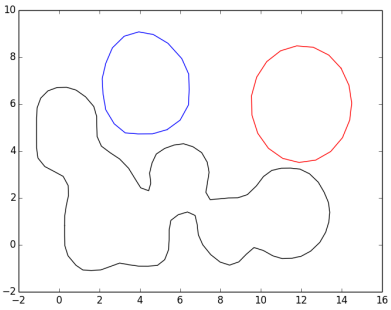
(e) Experiment 4 Initial Samples.



(f) Experiment 4 Interpolation.



(g) Experiments 5, 6 and 7 Initial Samples.



(h) Experiments 5, 6 and 7 Interpolation.

Figure 4.1: Interpolation Experiment using all scenarios adopted in this work.

Experiment	Execution Time (s)
Experiment 1 and 3	$1.21 \cdot 10^{-1}$
Experiment 2	$1.12 \cdot 10^{-1}$
Experiment 4	$1.30 \cdot 10^{-1}$
Experiment 5, 6 and 7	$1.41 \cdot 10^{-1}$

Table 4.1: Execution time for*B-Spline* interpolation of all scenarios used in this work.

Another important experiment regarding the usage of *Basis Splines* is to check how does the interpolation behave with a reduced set of samples. Considering the same perimeters from the previous experiment, on Figure4.2 we show the result of the interpolation when an amount of 15% of the original samples are removed from the set prior to the interpolation.

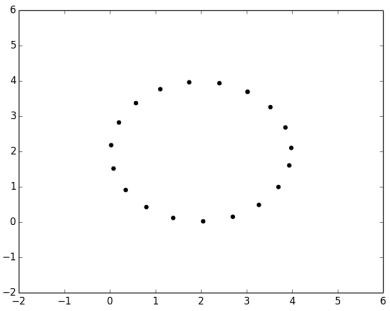
It is possible to see that the number of samples plays a major role when creating a *Basis Spline* representation. From the interpolated results – Figures 4.2b, 4.2d, 4.2f and 4.2h – due to the reduced number of samples all perimeters now present rough and sharp edges when compared to their analogous results – Figures 4.1b, 4.1d, 4.1f and 4.1h. This result is even more noticeable when considering shapes that are not similar to circles, as shown on Figure 4.2d and on the largest perimeter of Figure 4.2h.

Such phenomena would impact on the navigation of the vehicles given that this representation is more likely to represent a less accurate boundary. It also impacts the navigation given the fact that with a smaller number of samples, the smoothness of the continuous path is compromised, and despite the fact that we still have continuity in C^2 – thus the tangent and acceleration vectors still exists – it is most likely that we will observe a bigger tracking error on the x and y positions.

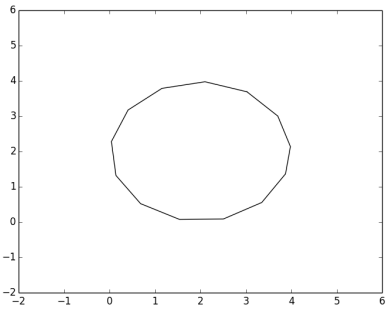
Nevertheless, reducing the number of samples also reduces the time to calculate the interpolation, as shown on Table 4.2 . However, as this time was already within an acceptable range in the context of this work, this reduction does not pose an improvement given the consequences of reducing the set of samples explained before.

Experiment	Execution Time (s)
Experiment 1 and 3	$0.83 \cdot 10^{-1}$
Experiment 2	$0.92 \cdot 10^{-1}$
Experiment 4	$0.97 \cdot 10^{-1}$
Experiment 5, 6 and 7	$1.01 \cdot 10^{-1}$

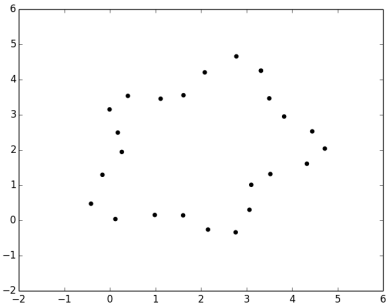
Table 4.2: Execution time for*B-Spline* interpolation of all scenarios used in this work with a reduced set of initial samples.



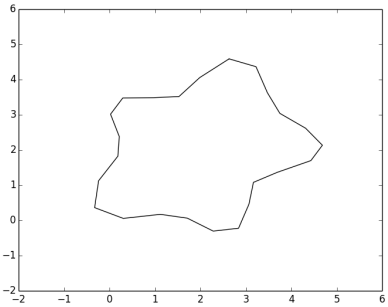
(a) Experiments 1 and 3 Reduced Set of Initial Samples.



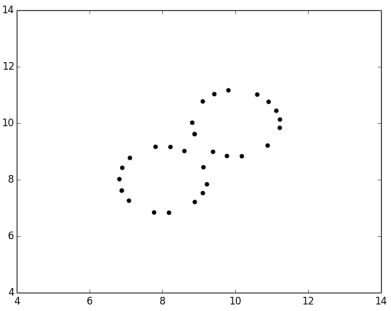
(b) Experiments 1 and 3 Interpolation.



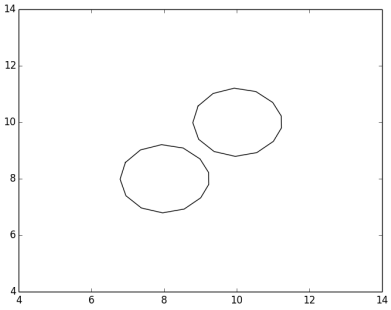
(c) Experiment 2 Initial Reduced Set of Samples.



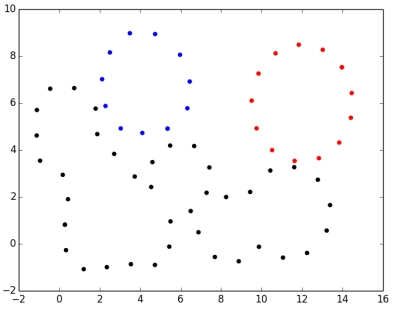
(d) Experiment 2 Initial Interpolation.



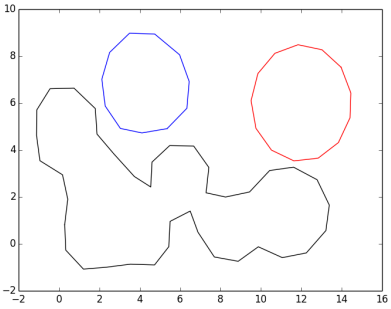
(e) Experiment 4 Initial Reduced Set of Samples.



(f) Experiment 4 Interpolation.



(g) Experiments 5, 6 and 7 Reduced Set of Initial Samples.

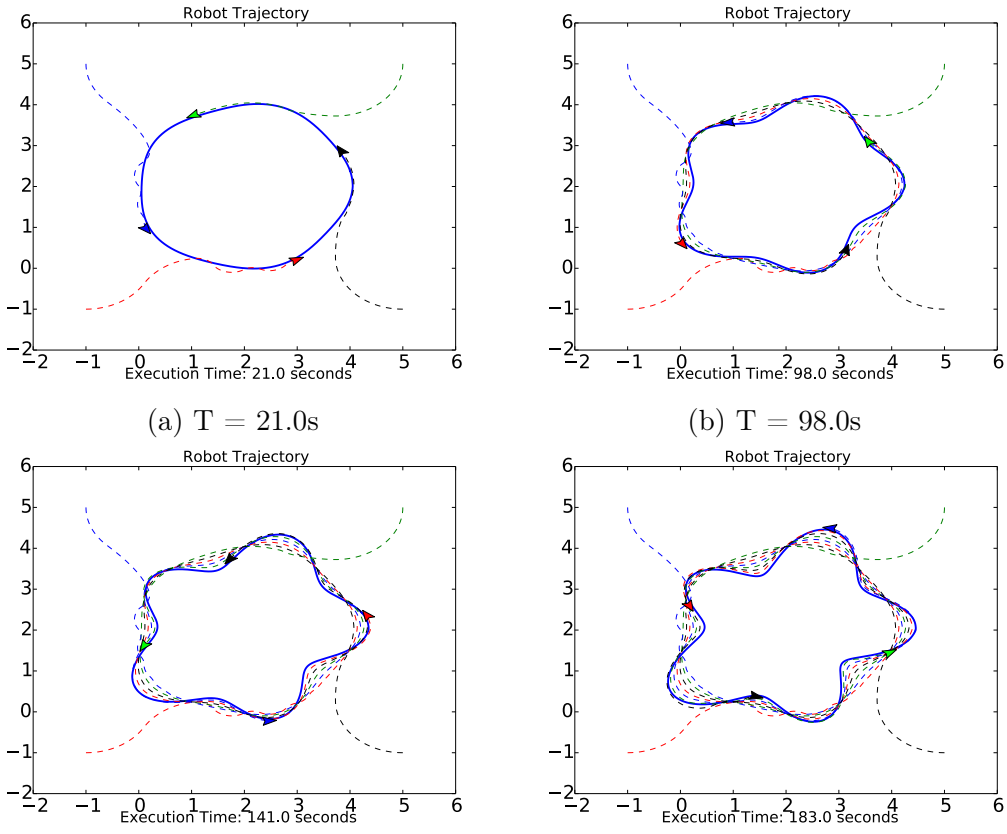


(h) Experiments 5, 6 and 7 Interpolation.

Figure 4.2: Interpolation Experiment using all scenarios adopted in this work.

4.2 Dynamic Perimeter Tracking

This experiment aims to validate our tracking strategy by evaluating the robot's actuators and tracking error when the perimeters are under severe perturbation. As can be seen in Figure 4.3, our navigation strategy not only does work but also does not depend on the original position of the vehicle. In this experiment, we show that even starting in opposed positions, multiple vehicles are capable of maintaining the tracking of the desired curve.



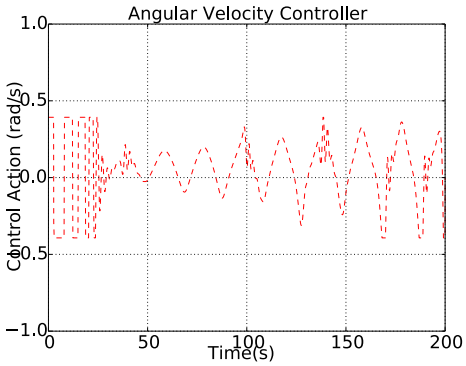
(a) T = 21.0s (b) T = 98.0s

(c) T = 141.0s (d) T = 183.0s

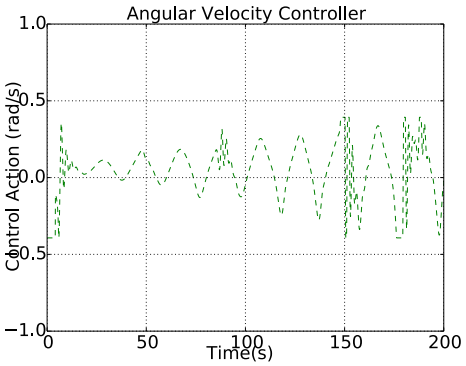
Figure 4.3: Experiment 1: Tracking a perimeter under severe perturbation.

As for the actuators - angular velocity and linear velocity - of those vehicles in this experiment, we detected the same behavior as for the single robot experiment. Figure 4.4 shows the control action values for the angular velocity actuator and linear velocity actuators for all the vehicles during the tracking of the perimeter.

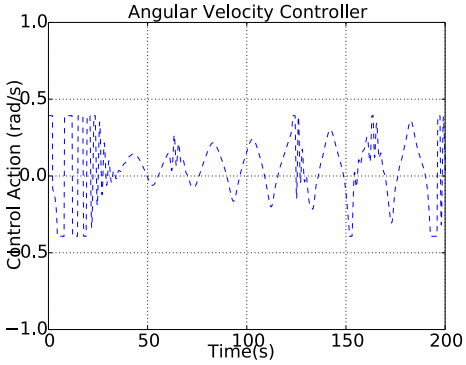
It is also important to notice that the tracking error for all vehicles also tends to zero, as they are capable of tracking the perimeter as it changes, as presented in Figure 4.5:



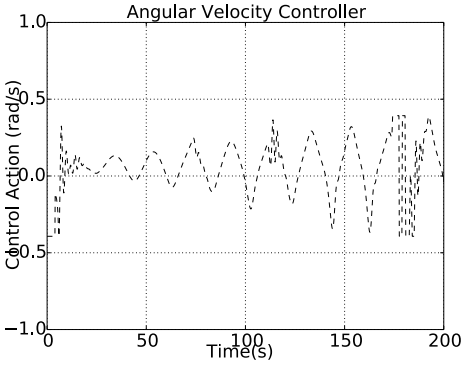
(a) Angular Velocity actuator of the red robot.



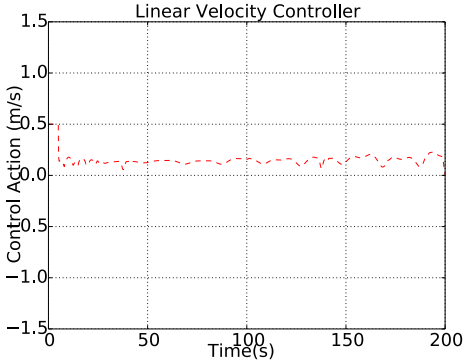
(b) Angular Velocity actuator of the green robot.



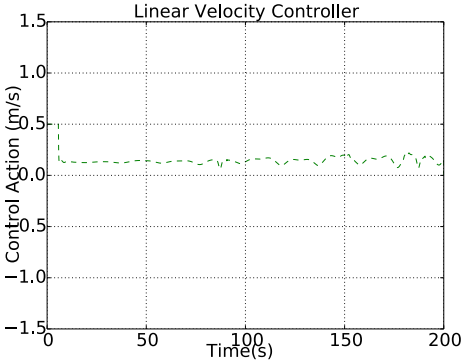
(c) Angular Velocity actuator of the blue robot.



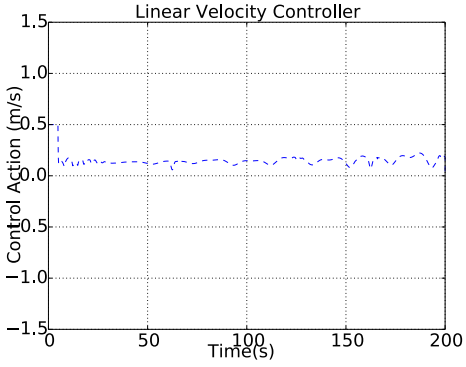
(d) Angular Velocity actuator of the black robot.



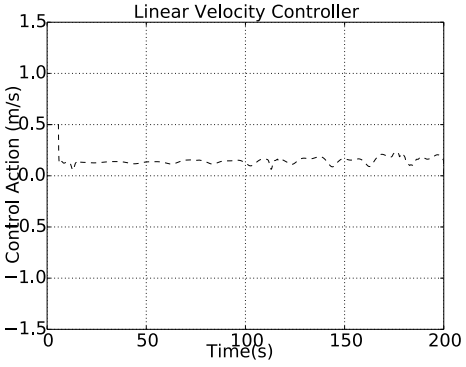
(e) Linear Velocity actuator of the red robot.



(f) Linear Velocity actuator of the green robot.

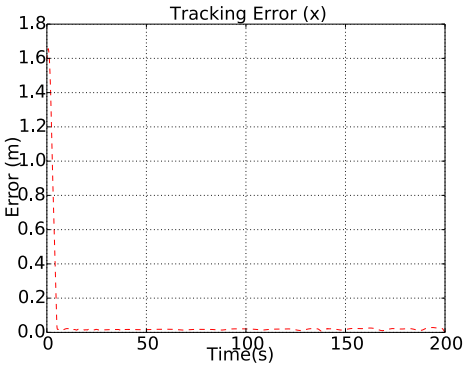


(g) Linear Velocity actuator of the blue robot.

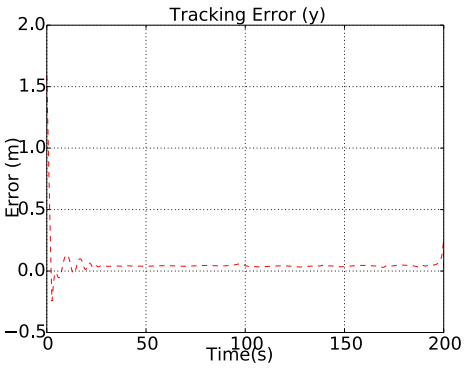


(h) Linear Velocity actuator of the black robot.

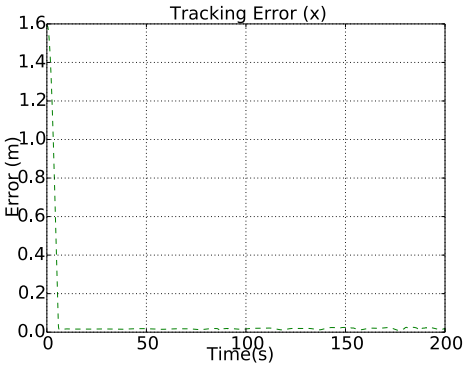
Figure 4.4: Experiment 1: Control action values for the angular velocity actuator and linear velocity actuator for all vehicles.



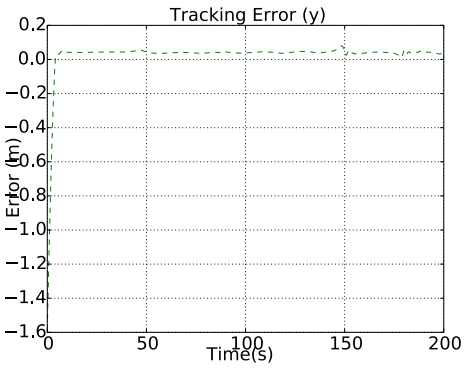
(a) Tracking Error in x of the red robot



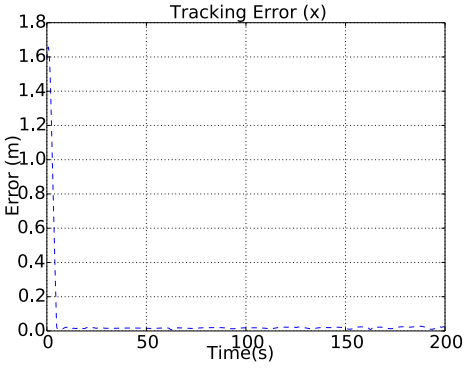
(b) Tracking Error in y of the red robot.



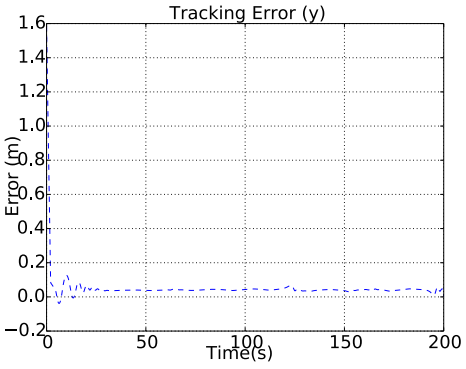
(c) Tracking Error in x of the green robot.



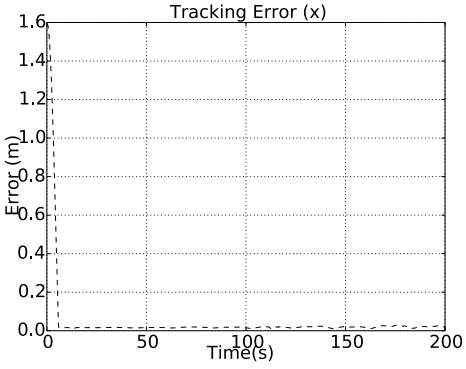
(d) Tracking Error in y of the green robot



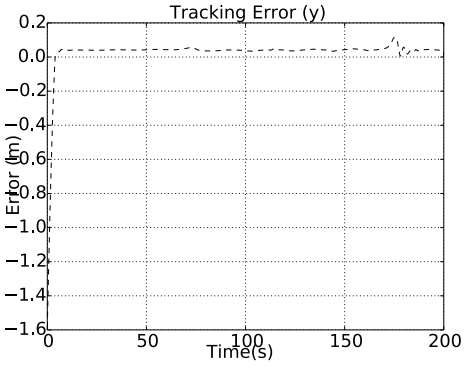
(e) Tracking Error in x of the blue robot.



(f) Tracking Error in y of the blue robot.



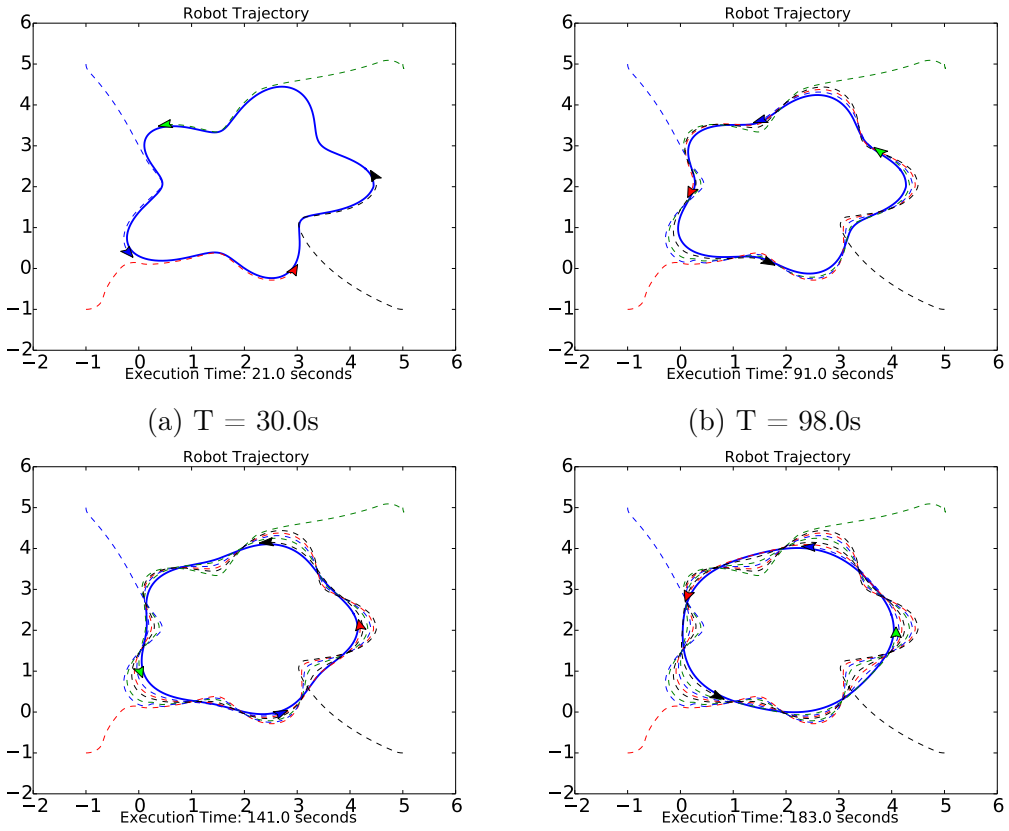
(g) Tracking Error in x of the black robot.



(h) Tracking Error in y of the black robot.

Figure 4.5: Experiment 1: Tracking errors in x and y for all vehicles.

Our strategy not only relies on the fact that perimeters are always turning into a more complex shape but also on the fact that perimeters can become a simple one. In the following experiment, we show the opposite scenario of the former experiment, in which multiple vehicles are capable of tracking the polygon as it turns from a star-like shape into a simple circle. As can be seen in Figure 4.6 the vehicles are capable of maintaining the tracking.



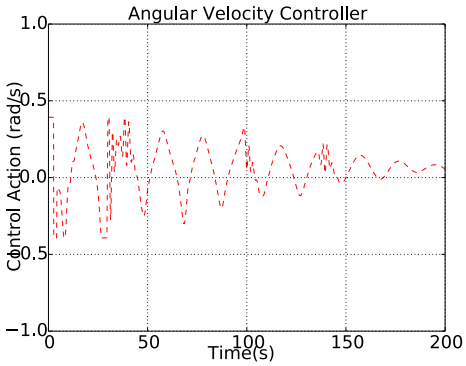
(a) T = 30.0s (b) T = 98.0s

(c) T = 137.0s (d) T = 191.0s

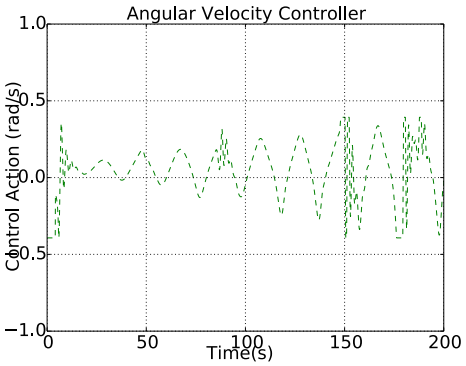
Figure 4.6: Experiment 2: Severe perturbation scenario with multiple agents tracking it as it goes in the opposite direction of the former experiment. The solid blue line is the perimeter, and the dashed line is the robot's trajectory according to its triangle color, which is represented by a colored triangle.

As for the actuators, again Linear and Angular velocity controllers, Figure 4.7 and show the values for both actuators. Again, it is possible to identify oscillations on the angular velocity controller, but always within the possible range.

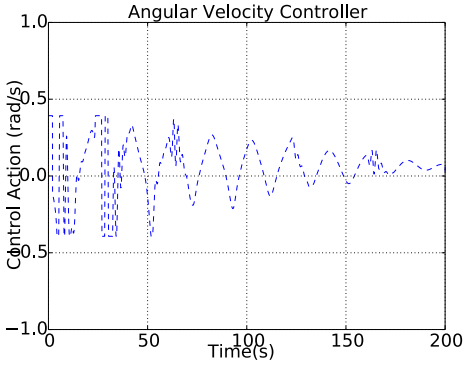
Finally, once again we show that our strategy tends to lead the tracking error to zero by showing that in both x and y axis the errors for both vehicles tend to go to zero as they approach the perimeter to be tracked and maintain tracking it, as can be seen in Figure 4.8.



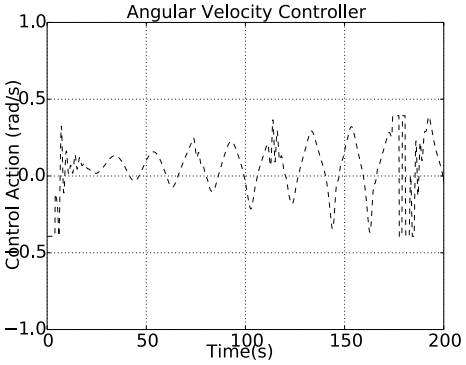
(a) Angular Velocity actuator of the red robot.



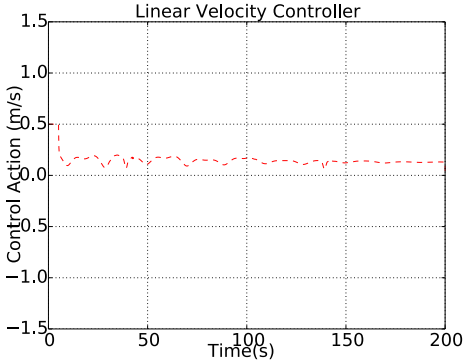
(b) Angular Velocity actuator of the green robot.



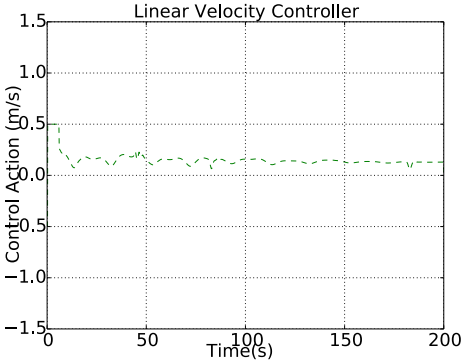
(c) Angular Velocity actuator of the blue robot.



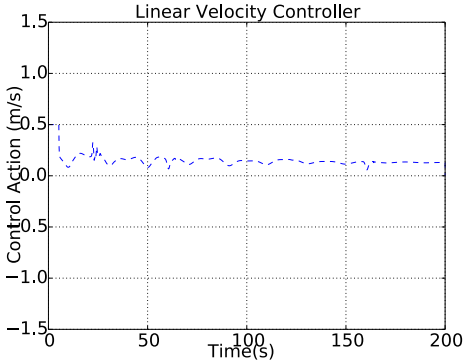
(d) Angular Velocity actuator of the black robot.



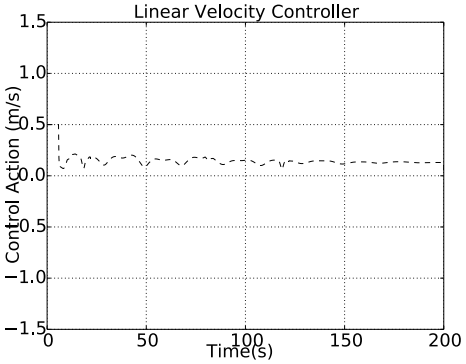
(e) Linear Velocity actuator of the red robot.



(f) Linear Velocity actuator of the green robot.

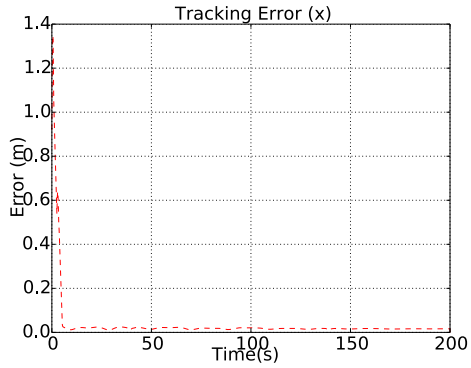


(g) Linear Velocity actuator of the blue robot.

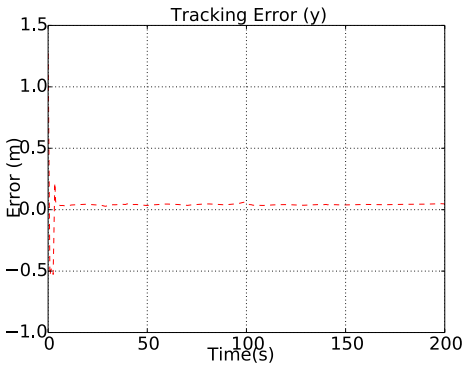


(h) Linear Velocity actuator of the black robot.

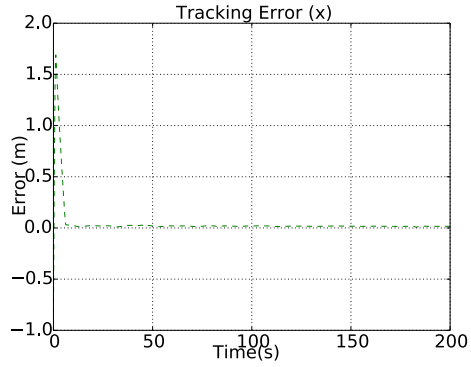
Figure 4.7: Experiment 2: Control action values for the angular velocity actuator and linear velocity actuator for all vehicles.



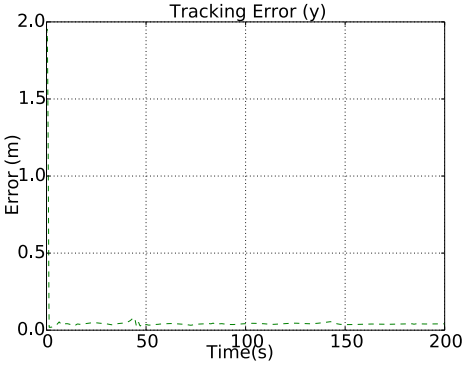
(a) Tracking Error in x of the red robot



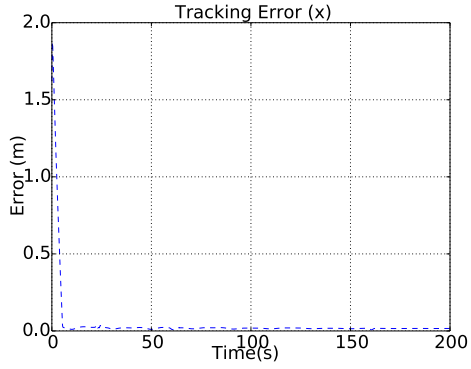
(b) Tracking Error in y of the red robot.



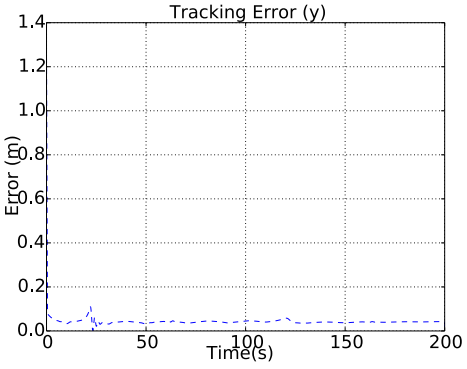
(c) Tracking Error in x of the green robot.



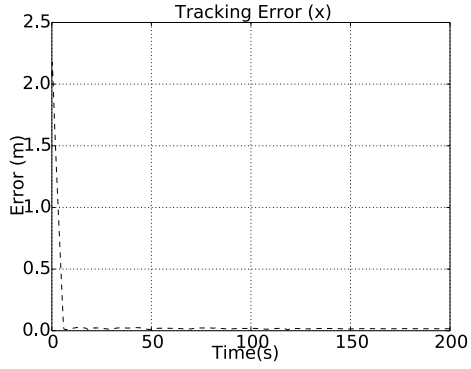
(d) Tracking Error in y of the green robot



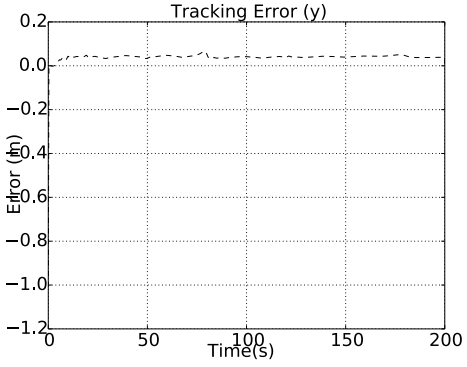
(e) Tracking Error in x of the blue robot.



(f) Tracking Error in y of the blue robot.



(g) Tracking Error in x of the black robot.

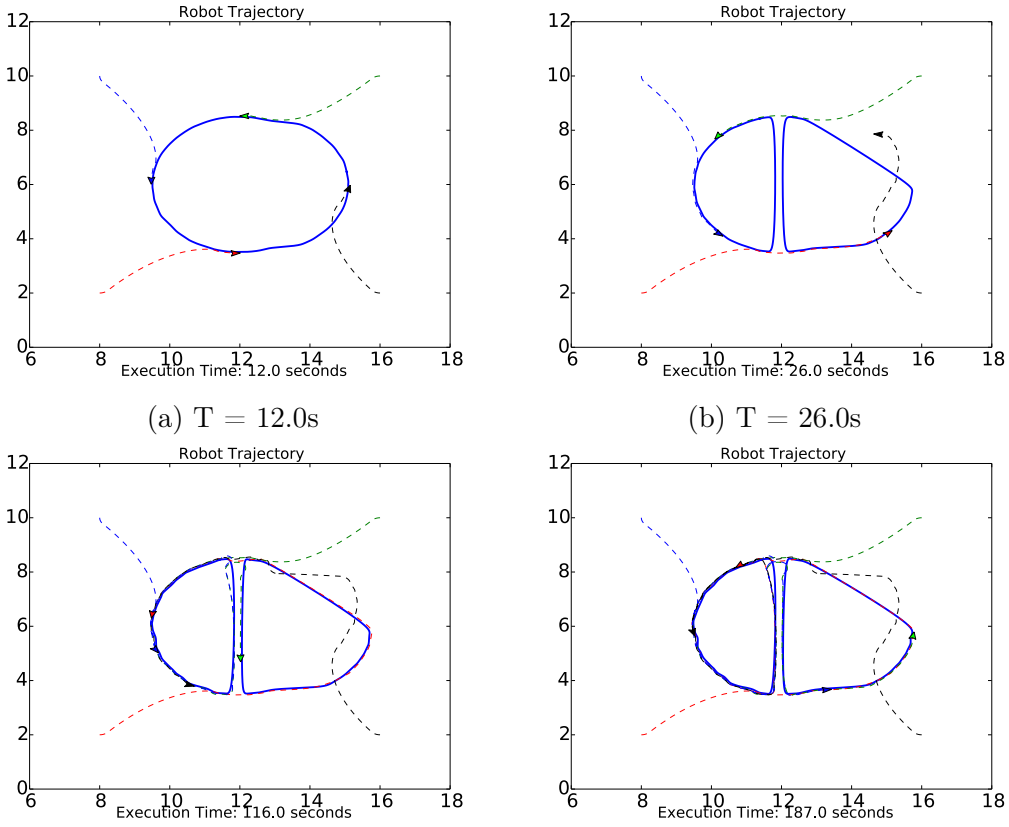


(h) Tracking Error in y of the black robot.

Figure 4.8: Experiment 2: Tracking errors in x and y for all vehicles.

4.3 Split Perimeters Tracking

In this experiment, we aim to study the behavior of the robot in a scenario in which a split would occur. For this experiment, we start with a single perimeter that resembles a circle and deforms itself resembling an ellipsis-like form, as shown in Figure 4.9a. We determine an unknown behavior for the perimeter's samples, which means that there is no mathematical model that can predict the dynamics of the perimeter in order to be incorporated into the vehicle's dynamics. Also, we guarantee that property defined by Equation (3.2) is valid. A perimeter split situation is achieved approximately to second 26, as shown on Figure 4.9b:



(a) T = 12.0s (b) T = 26.0s

(c) T = 116.0s (d) T = 183.0s

Figure 4.9: Experiment 3: Perimeter Split scenario

After the perimeter split occurs, it is possible to analyze the behavior of each robot individually. Starting with the Red robot, this vehicle maintains the tracking of its former perimeter. This is a completely understandable behavior as the rightmost perimeter is the one with a larger perimeter and the robot is already located on its boundaries, which makes the traveling time (t_t) tend to zero. Therefore, the *coverage efficiency* metric for this perimeter is certainly higher than the same metric for the

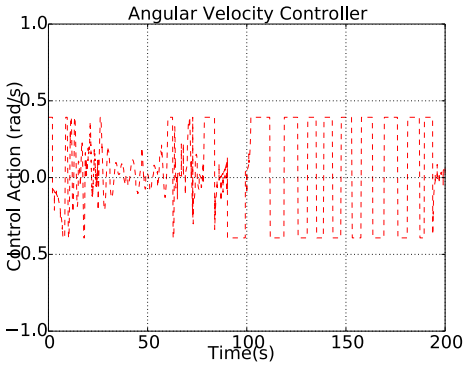
leftmost perimeter. As it approaches the leftmost perimeter, it swaps to it, as the value of the coverage efficiency for the leftmost perimeter becomes bigger due to the decrease of the distance of the Red robot to this perimeter.

As for the Green robot, it is possible to see that it maintains its course on the leftmost perimeter but as soon as he reaches the closest exit point, it goes to the rightmost perimeter. Despite the fact that the traveling time (t_t) for this vehicle when it comes to the leftmost perimeter tends to zero, as the vehicle is already on the curve, it is possible to identify that the rightmost perimeter has a coverage time (c_t) which is substantially bigger than the leftmost perimeter. Also, the traveling time (t_t) for the rightmost becomes smaller and smaller as the vehicle approaches the exit points, which makes it moves to the rightmost perimeter as soon as it reaches this exit point. When the robot changes from one perimeter to another, it is a confirmation that our allocation strategy works as expected.

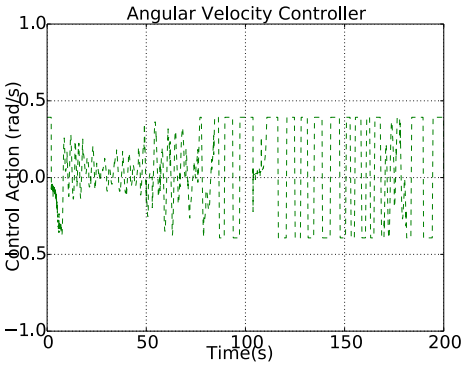
The Blue robot maintains the same pattern as the one observed on the Green robot: it keeps tracking the leftmost curve – the one in which he was previous to the split – but as it approaches the closest exit point to the rightmost perimeter, the vehicle turns to the rightmost curve and starts tracking it.

Finally, the black robot is the one with the most different behavior. As the split occurs, the black robot finds itself not tracking any of the perimeters: samples that previously represented the curve that it was following are gone as the perimeter collapsed. Therefore, the robot finds the closest sample that it must track, which is located closer to the left top corner of the rightmost perimeter. It is also important to notice that this is the exit point of the rightmost perimeter. Therefore the black robot briefly tracks the rightmost perimeter but as soon as it reaches the exit point it moves to the leftmost perimeter and stats tracking it. The robot remains to track this perimeter until the end of the simulation, as the Blue vehicle moves to the rightmost behavior and finally we reach a situation in which the number of robots is equally divided on the perimeters: two for each.

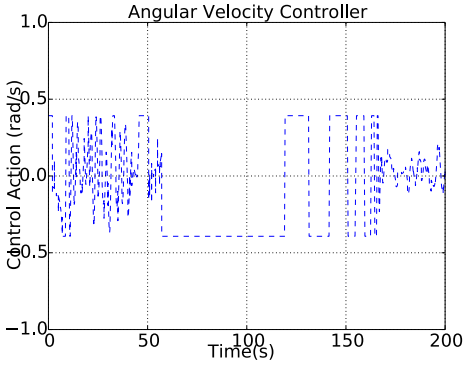
Once the behavior of each robot was explained it is important to evaluate the value of the control action on their actuators and also the tracking error for each of them. The control action on the angular and linear velocity controllers for each of the vehicles is shown in Figure 4.10 whilst the tracking errors in x and y also for each vehicle are depicted on Figure 4.11.



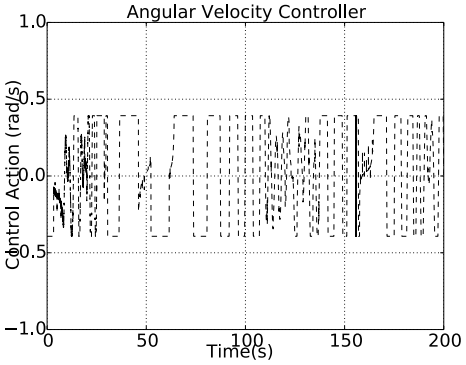
(a) Angular Velocity actuator of the red robot.



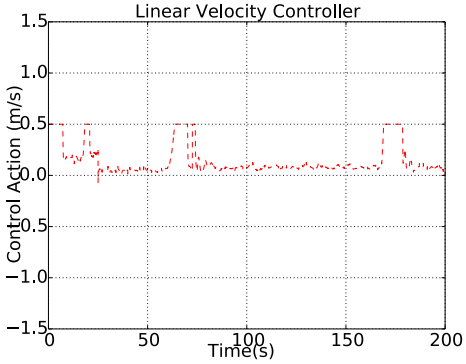
(b) Angular Velocity actuator of the green robot.



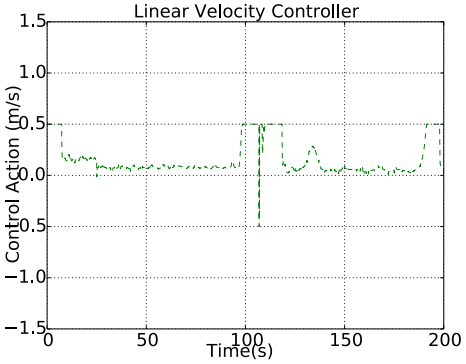
(c) Angular Velocity actuator of the blue robot.



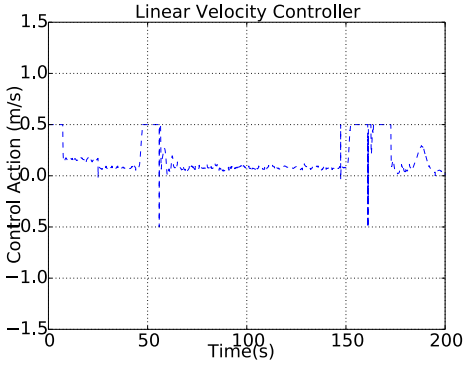
(d) Angular Velocity actuator of the black robot.



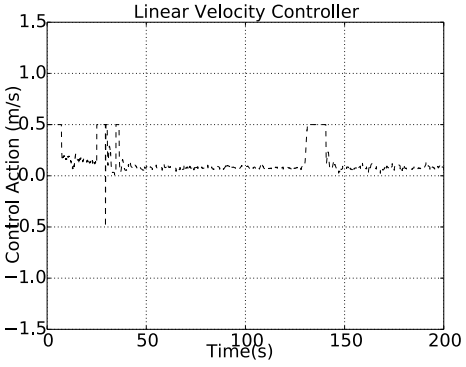
(e) Linear Velocity actuator of the red robot.



(f) Linear Velocity actuator of the green robot.

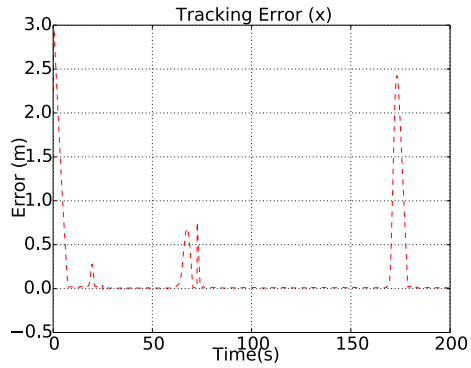


(g) Linear Velocity actuator of the blue robot.

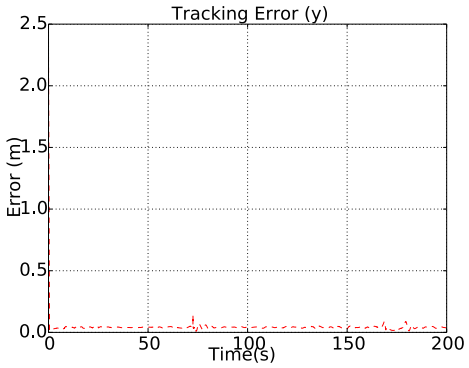


(h) Linear Velocity actuator of the black robot.

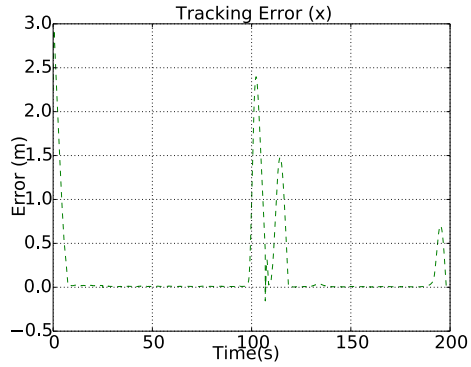
Figure 4.10: Experiment 3: Control action values for the angular velocity actuator and linear velocity actuator for all vehicles.



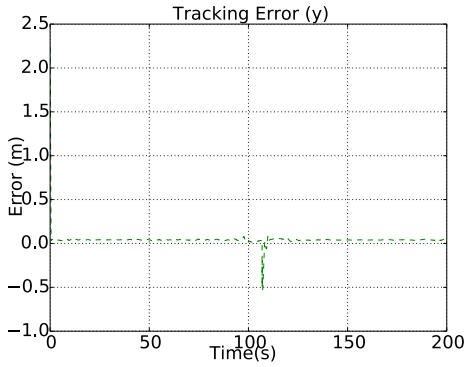
(a) Tracking Error in x of the red robot



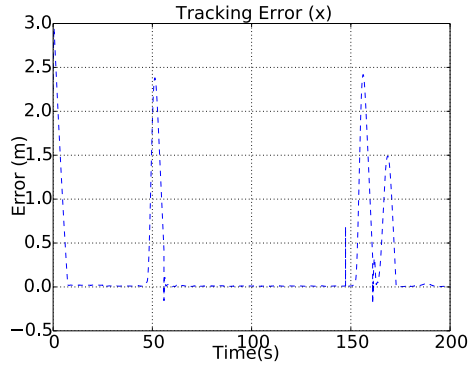
(b) Tracking Error in y of the red robot.



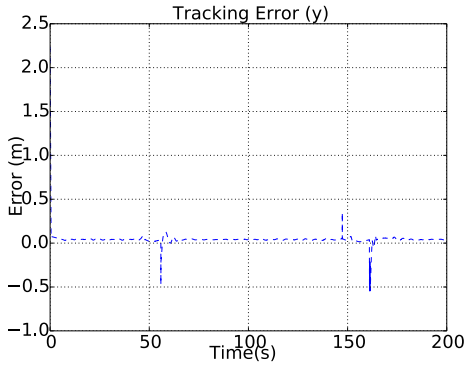
(c) Tracking Error in x of the green robot.



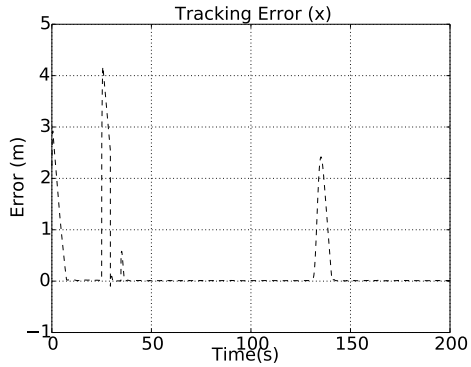
(d) Tracking Error in y of the green robot



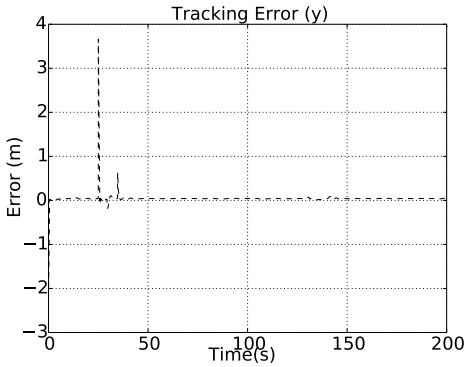
(e) Tracking Error in x of the blue robot.



(f) Tracking Error in y of the blue robot.



(g) Tracking Error in x of the black robot.



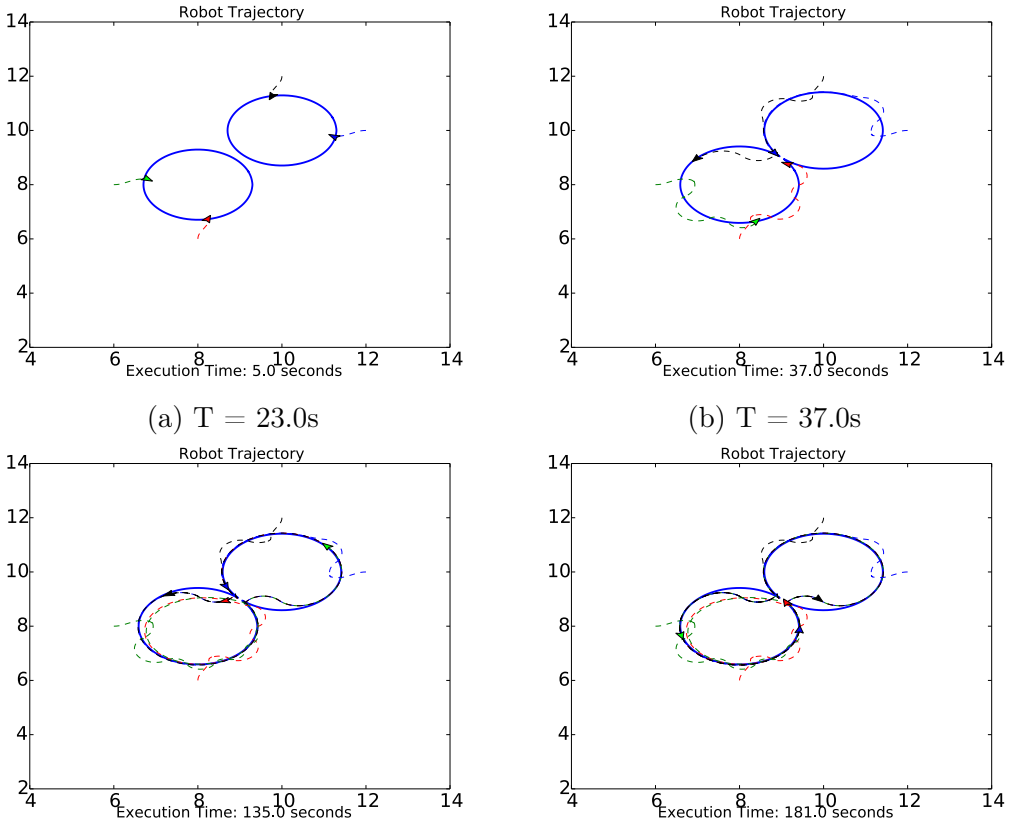
(h) Tracking Error in y of the black robot.

Figure 4.11: Experiment 3: Tracking errors in x and y for all vehicles.

4.4 Merging Perimeters Tracking

Finally, another situation that may happen while the robots are navigating is to two or more perimeters merge. This happens when the boundaries of one perimeter is too close to the boundaries of the other, leaving no space for the vehicle to pass without going into one of them. In this case, those perimeters are merged into one. What is more likely to happen is that the vehicle will keep on tracking the perimeter that has merged due to the increase of the total area – hence the value for the estimated coverage time will increase as the perimeter is now larger and the traveling time will tend to zero as the robot is already located on the curve.

In this experiment, we explore the behavior of multiple agents when the curve that they are following, represented in this scenario by two circles that are increasing their radius at the same rate, merges and form a new shape that resembles a hippopede. Figure 4.12 shows the result of this experiment.



(a) T = 23.0s (b) T = 37.0s

(c) T = 90.0s (d) T = 146.0s

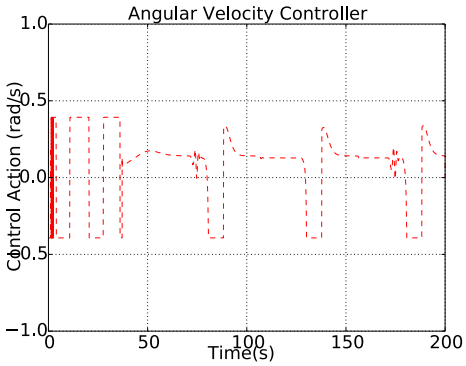
Figure 4.12: Experiment 4: Merging of two different perimeters.

In the previous experiment, it is possible to see that after merging with the other perimeter – as shown in Figure 4.12b – the section that merged forms an extremely narrow passage. Therefore, all the vehicles find difficulties following the perimeter on that section due to the limitations on their angular velocity – a limitation that we added so the vehicles can represent some sort of actual vehicle, point being that the angular velocity is saturated at $\frac{\pi}{8}$. Overall, when passing through that part of the perimeter, all vehicles enter the perimeter zone and then follow it correctly.

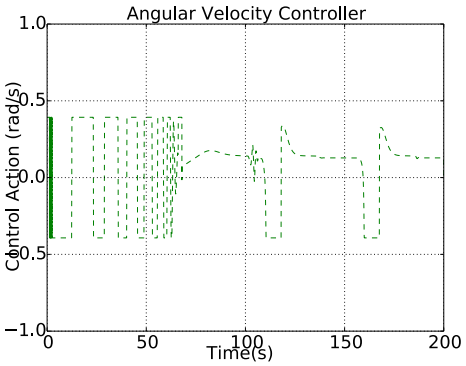
Analyzing each vehicle, it is possible to identify that the behavior all four of them is similar: before merging they all follow their respective perimeter which is expanding and become a circle with a larger radius. After both circles merging, all vehicles keep on tracking both curves almost the same way. It is possible to identify that they even make the same route once the merge occurs, and also that in order to pass through that narrow region, they enter the perimeter to be able to follow it.

It is important to notice, though, that the Red vehicle prior to following the curve after the merge, is pushed down by the blue robot. This happens due to the fact that both vehicles are extremely close – as shown in Figure 4.12b – and the potential fields technique prevent a collision between them which would likely occur. The Red vehicle then goes inside the perimeter and starts to get distance from the blue vehicle at the same time that it tries to reach the boundaries of the perimeter. Once it is tracking the perimeter again, the Red vehicle goes follows the group pattern and all vehicles are back on track the whole merged perimeter.

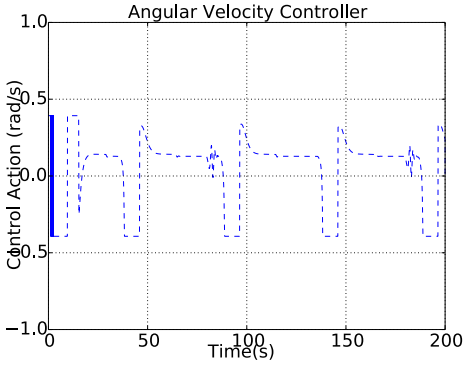
As the behavior of each robot was explained it is important to evaluate the value of the control action on their actuators and also the tracking error for each of them. The control action on the angular and linear velocity controllers for each of the vehicles is shown in Figure 4.13 whilst the tracking errors in x and y also for each vehicle are depicted on Figure 4.14.



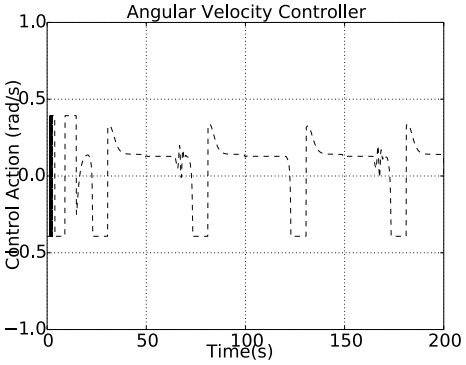
(a) Angular Velocity actuator of the red robot.



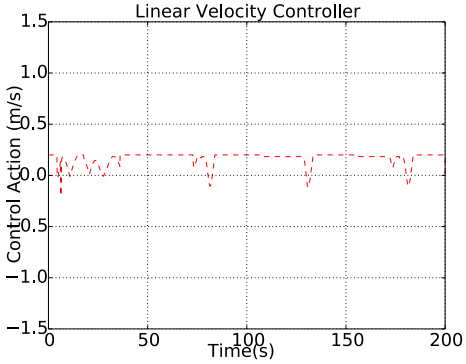
(b) Angular Velocity actuator of the green robot.



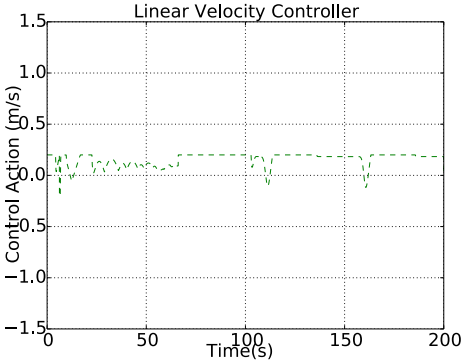
(c) Angular Velocity actuator of the blue robot.



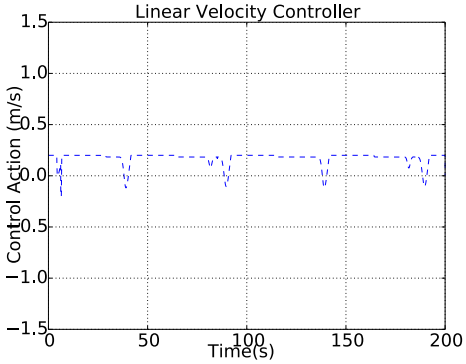
(d) Angular Velocity actuator of the black robot.



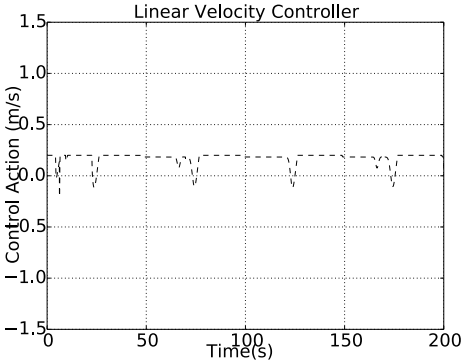
(e) Linear Velocity actuator of the red robot.



(f) Linear Velocity actuator of the green robot.

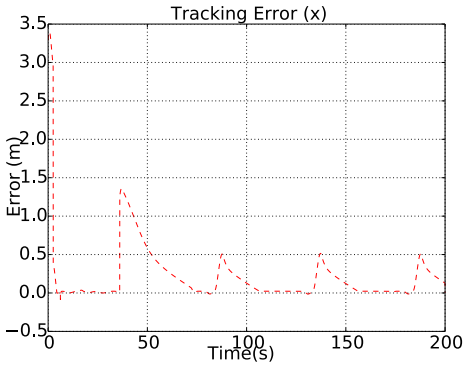


(g) Linear Velocity actuator of the blue robot.

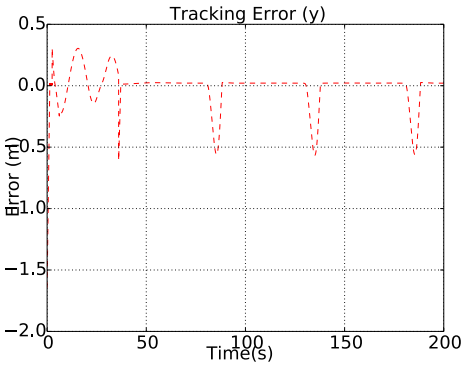


(h) Linear Velocity actuator of the black robot.

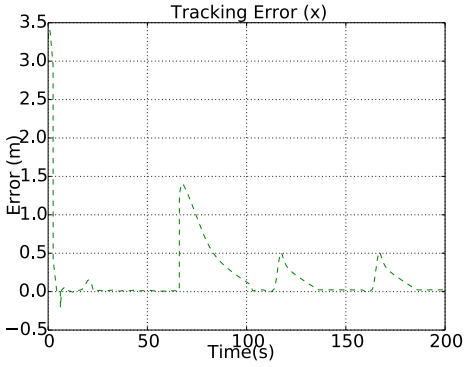
Figure 4.13: Experiment 4: Control action values for the angular velocity actuator and linear velocity actuator for all vehicles.



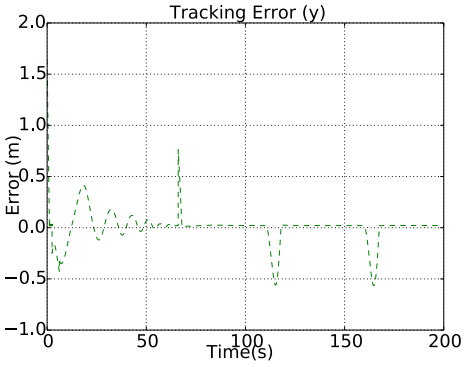
(a) Tracking Error in x of the red robot



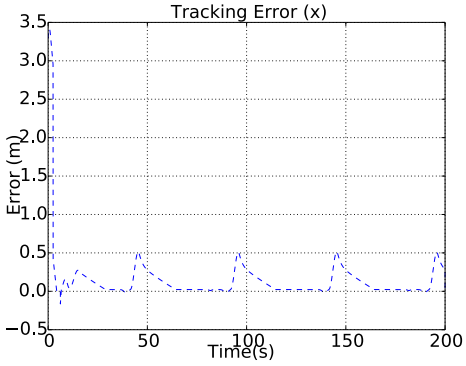
(b) Tracking Error in y of the red robot.



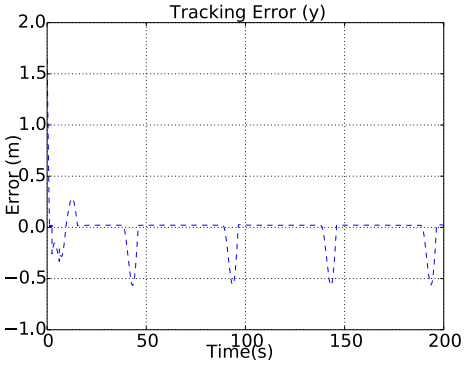
(c) Tracking Error in x of the green robot.



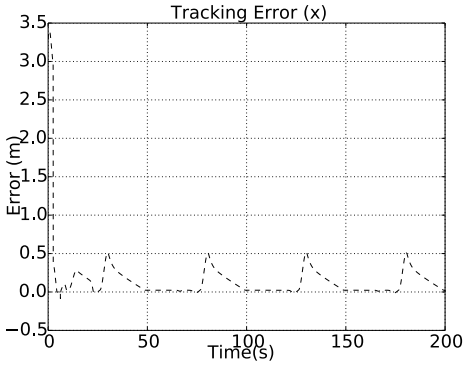
(d) Tracking Error in y of the green robot



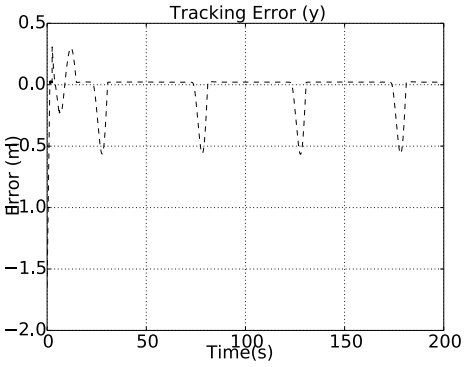
(e) Tracking Error in x of the blue robot.



(f) Tracking Error in y of the blue robot.



(g) Tracking Error in x of the black robot.



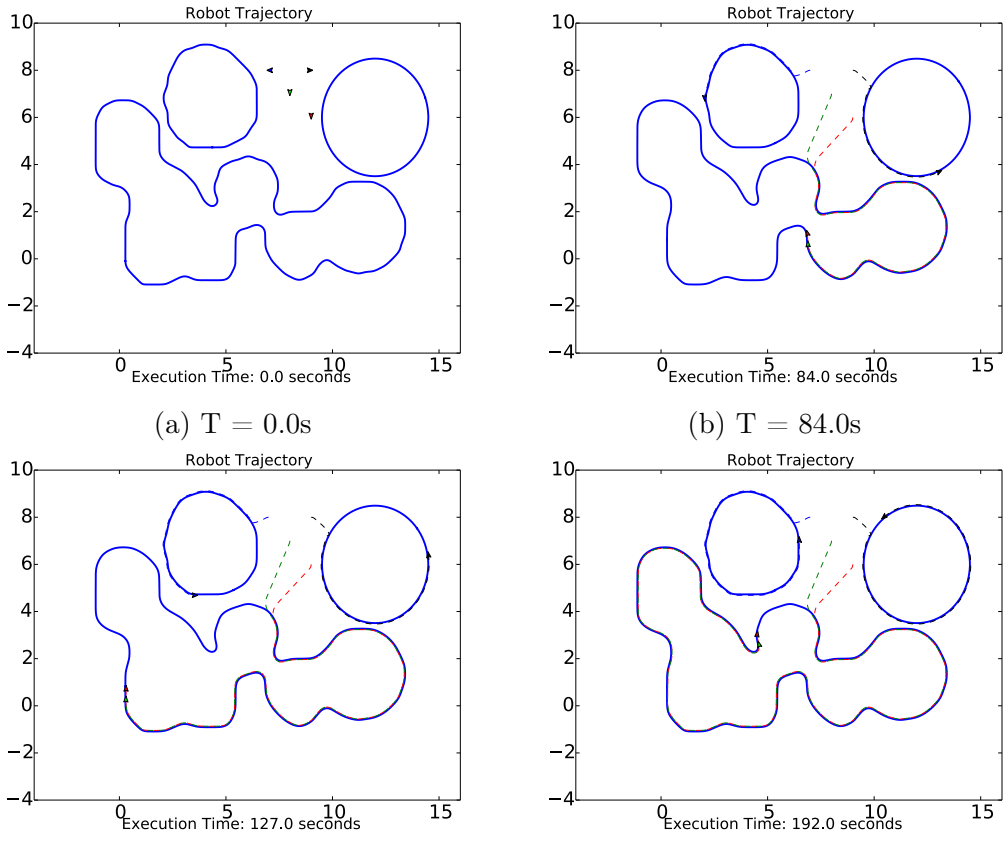
(h) Tracking Error in y of the black robot.

Figure 4.14: Experiment 4: Tracking errors in x and y for all vehicles.

4.5 Multiple Perimeters Tracking

In this section, we focus on an experiment involving multiple perimeters that must be tracked by a team of robots. In this scenario, a team of four robots has to patrol a region containing three different perimeters. All robots start close to each other – as shown in Figure 4.15a – and are allocated to those regions based on the coverage efficiency metric for each of them. Two robots are allocated to the area with the largest perimeter and the other two are allocated each to a remaining area.

The idea behind this experiment is to explore the robots as a team: since there are more robots (4) than the number of perimeters to cover (3), we focus on taking advantage of our allocation strategy to put more robots covering the largest perimeter whilst the others remain guarded by a single robot in a way that all regions are covered by at least one vehicle. Figure 4.15 shows the result of such experiment:



(a) T = 0.0s (b) T = 84.0s

(c) T = 127.0s (d) T = 192.0s

Figure 4.15: Experiment 5: Multiple vehicles being allocated to different regions and tracking them.

As can be seen, a pair of vehicles – Red and Green – are assigned to perform a perimeter track on the largest perimeter, whilst the other two remaining vehicles –

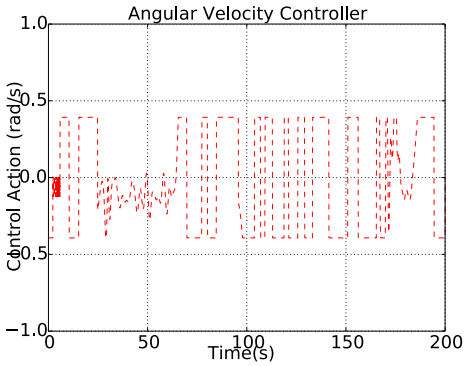
Blue and Black – are assigned to the remaining perimeters. It is also possible to verify that the Red and the Green vehicles are assigned to a way more complex perimeter with many more curves including a narrow one on the bottom. Despite all those complications, the vehicles maintained the capability of tracking the perimeter in a really effective way, staying on its boundaries every time. Also, it is also possible to see that due to the potential fields technique the vehicles always maintain a distance between then and do never collide, even when passing through those difficult parts of the perimeter.

When it comes to the Black vehicle, it is possible to see that it follows the perimeter which was assigned to it perfectly. Its perimeter – a simple circle – proved to be a really easy shape to be followed, with the Black vehicle following its contour without any problems and also without going inside the perimeter.

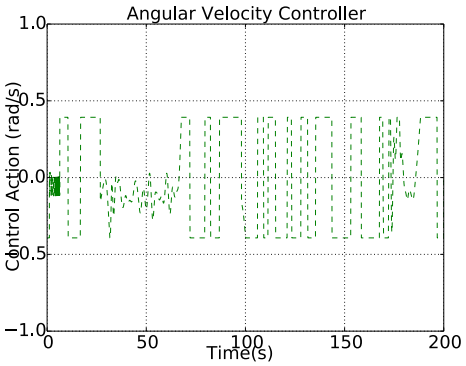
The Blue vehicle is also allocated to a really simple perimeter. Its shape resembles a circle, with some rough edges, but overall the vehicle followed it perfectly, and like the Black vehicle, not even once stepping inside the perimeter.

Finally, with this experiment and considering this kind of scenario with more vehicles to do the perimeter tracking than perimeters itself, it is safe to assume that when it comes to the allocation, the resources were allocated obeying two important characteristics of this problem: all perimeters were covered by at least one vehicle, and the largest perimeter received more vehicles. Those two characteristics are important for this work because they do corroborate one of coverage efficiency's principles which is the fact that in scenarios with more vehicles than perimeters, all perimeters will have at least one vehicle patrolling it.

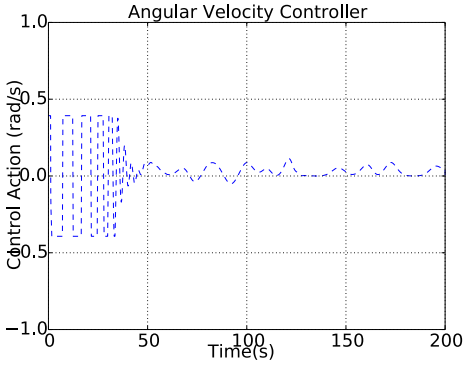
With the behavior of each vehicle analyzed and explained it is important to evaluate the value of the control action on their actuators and also the tracking error for each of them, especially considering the Red and Green robots which were under a really difficult perimeter to track. The control action on the angular and linear velocity controllers for each of the vehicles is shown in Figure 4.16 whilst the tracking errors in x and y also for each vehicle are depicted on Figure 4.17.



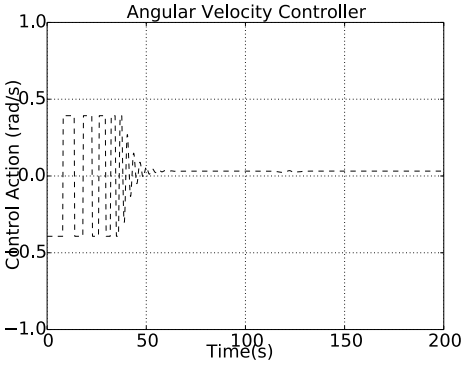
(a) Angular Velocity actuator of the red robot.



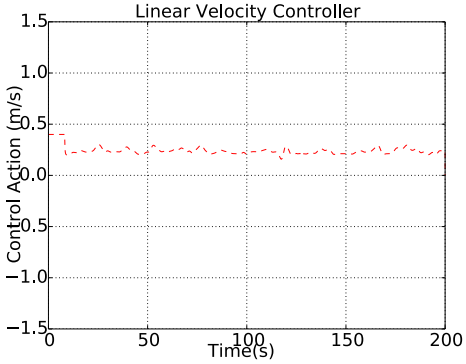
(b) Angular Velocity actuator of the green robot.



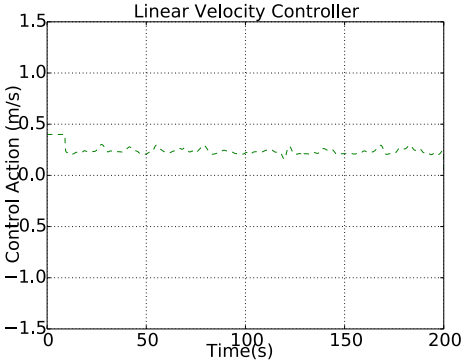
(c) Angular Velocity actuator of the blue robot.



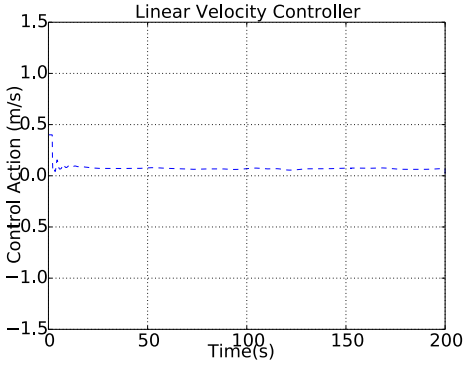
(d) Angular Velocity actuator of the black robot.



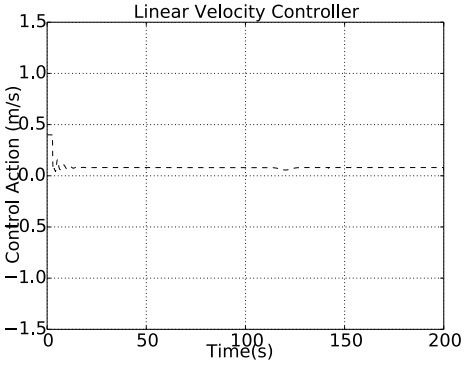
(e) Linear Velocity actuator of the red robot.



(f) Linear Velocity actuator of the green robot.

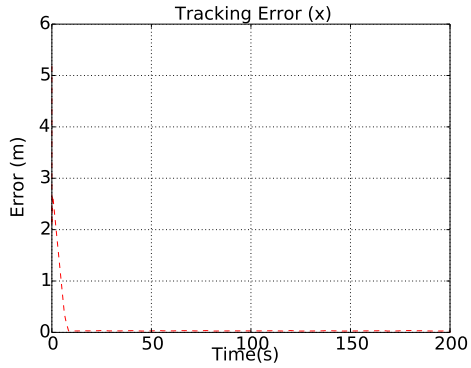


(g) Linear Velocity actuator of the blue robot.

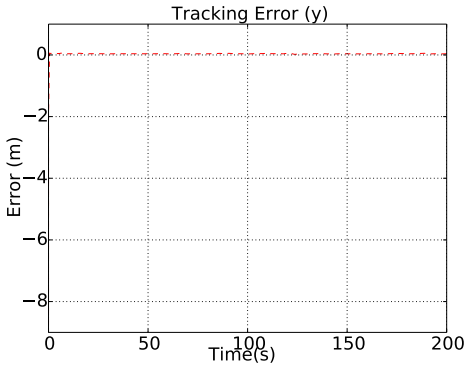


(h) Linear Velocity actuator of the black robot.

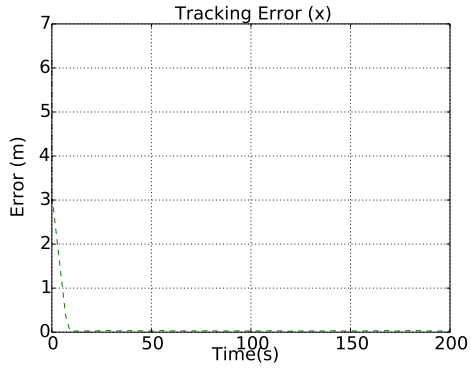
Figure 4.16: Experiment 5: Control action values for the angular velocity actuator and linear velocity actuator for all vehicles.



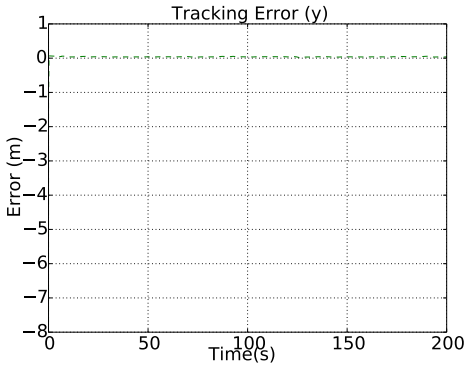
(a) Tracking Error in x of the red robot



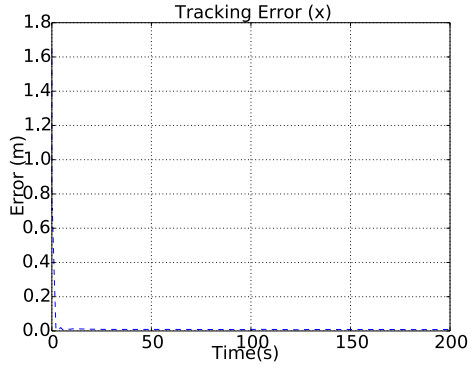
(b) Tracking Error in y of the red robot.



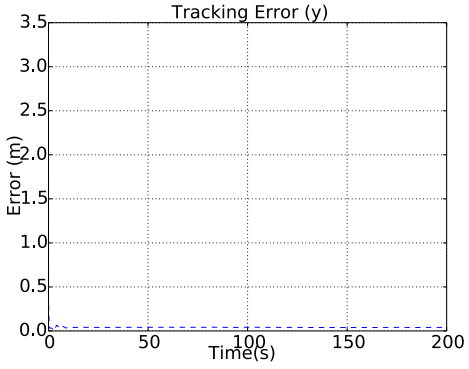
(c) Tracking Error in x of the green robot.



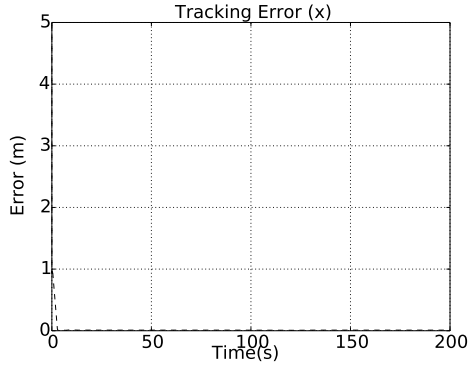
(d) Tracking Error in y of the green robot



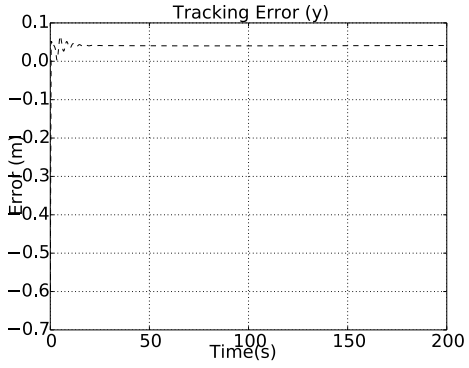
(e) Tracking Error in x of the blue robot.



(f) Tracking Error in y of the blue robot.



(g) Tracking Error in x of the black robot.

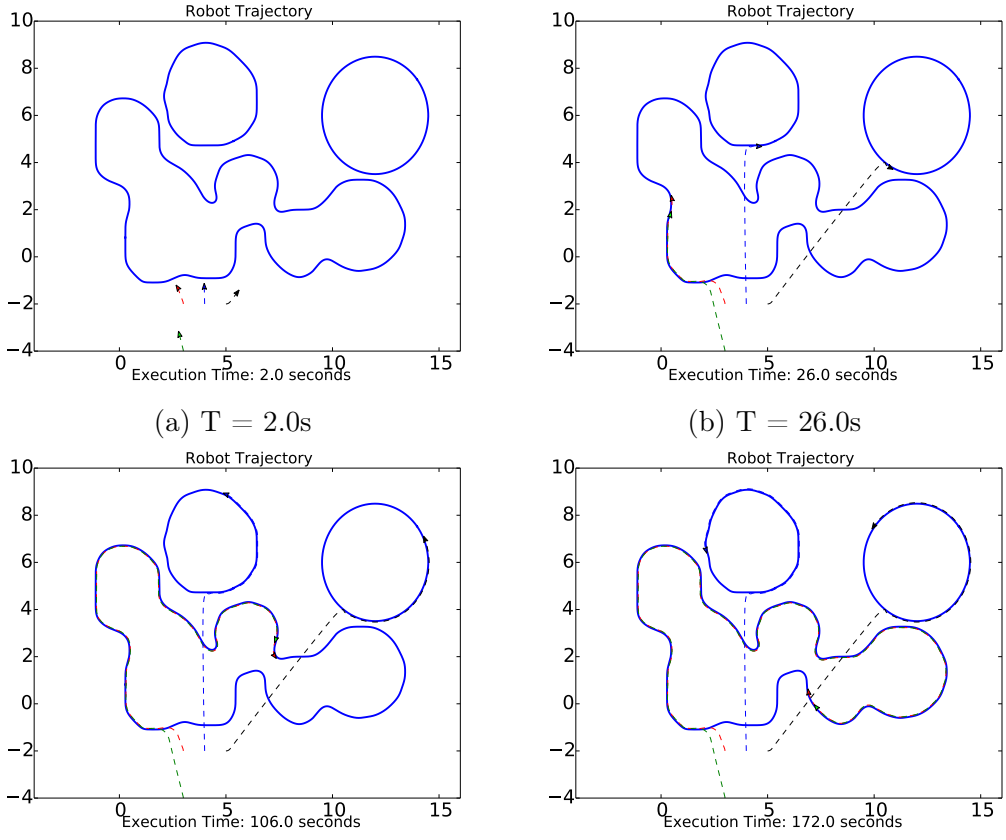


(h) Tracking Error in y of the black robot.

Figure 4.17: Experiment 5: Tracking errors in x and y for all vehicles.

Another experiment considering this same scenario was done, but now the team of vehicles is initially located at the bottom-most part of the scene. In this case, the only thing that changes is the fact that the Blue and the Black robot, in order to arrive at their designated perimeter, move through the largest perimeter. This is obviously an undesired situation since the robots who are traveling through a perimeter are exposed to its hazards. Still, our navigation and allocation algorithms are designed in a way that it does not take into account the fact that the vehicle can spend some time crossing a zone that can harm it. Therefore, the results of this experiment are valid and how to avoid dangerous zones is not on the scope of this work.

In this experiment, Red and Green robots are assigned to the largest perimeter area, whilst Black and Blue vehicles are assigned to the remaining perimeters. This scenarios is better described in Figure 4.18:

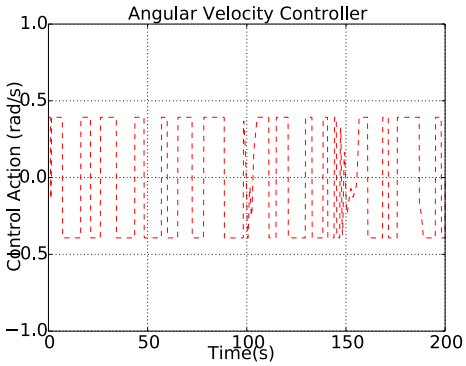


(a) T = 2.0s (b) T = 26.0s

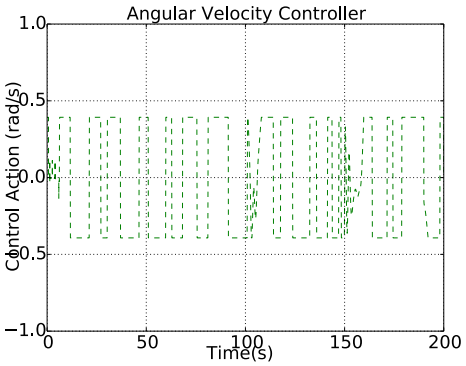
(c) T = 106.0s (d) T = 172.0s

Figure 4.18: Experiment 6: Multiple vehicles being allocated to different regions and tracking them starting from the bottom.

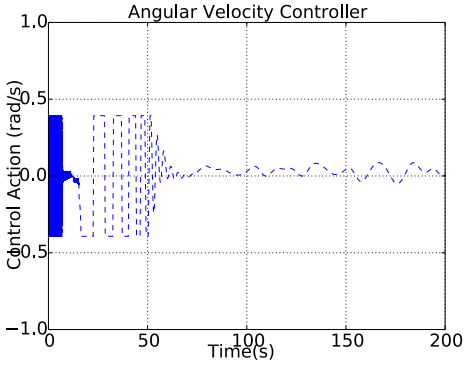
Once these results match the expected behavior of our algorithm, we must now analyze the behavior of the controllers and the tracking error of all vehicles. Both of those are described in Figures 4.19 and 4.20 respectively.



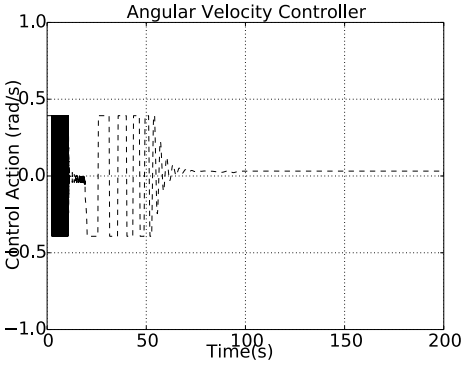
(a) Angular Velocity actuator of the red robot.



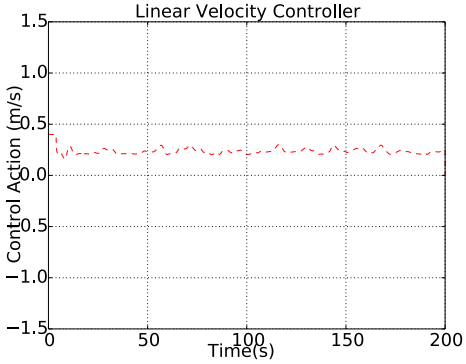
(b) Angular Velocity actuator of the green robot.



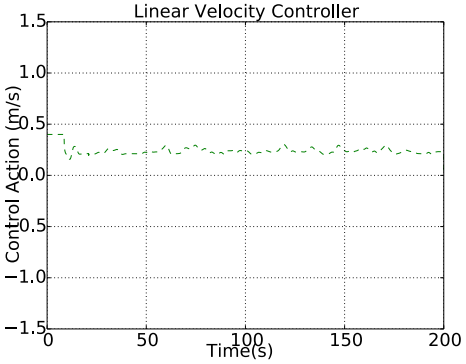
(c) Angular Velocity actuator of the blue robot.



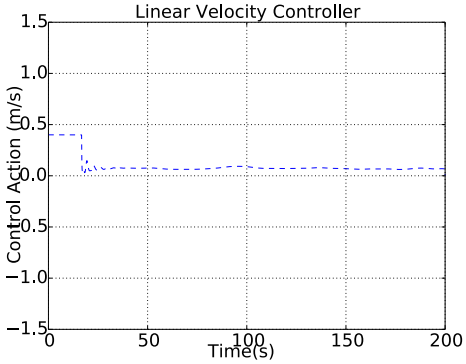
(d) Angular Velocity actuator of the black robot.



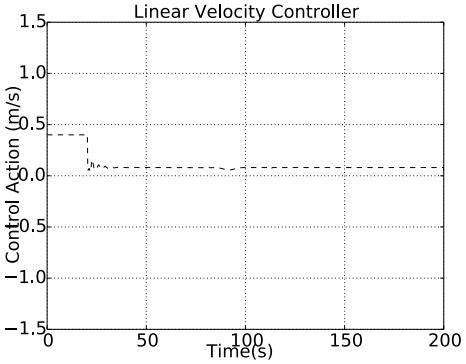
(e) Linear Velocity actuator of the red robot.



(f) Linear Velocity actuator of the green robot.

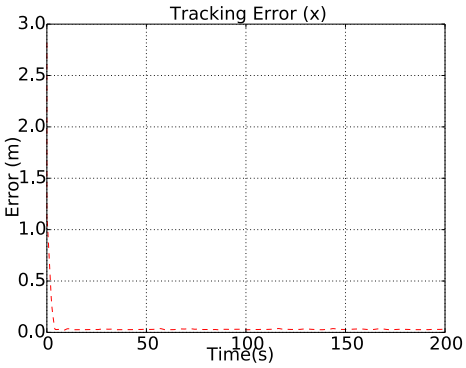


(g) Linear Velocity actuator of the blue robot.

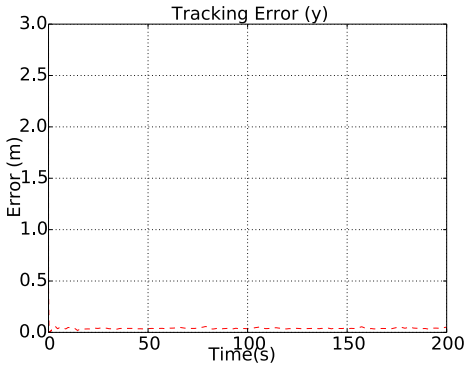


(h) Linear Velocity actuator of the black robot.

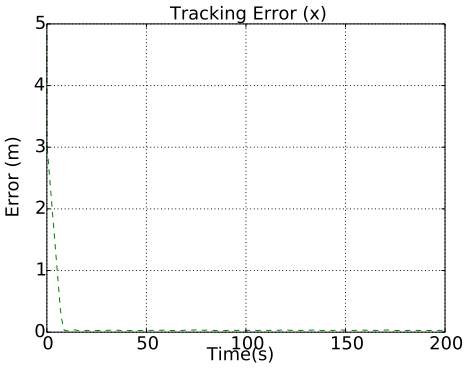
Figure 4.19: Experiment 6: Control action values for the angular velocity actuator and linear velocity actuator for all vehicles.



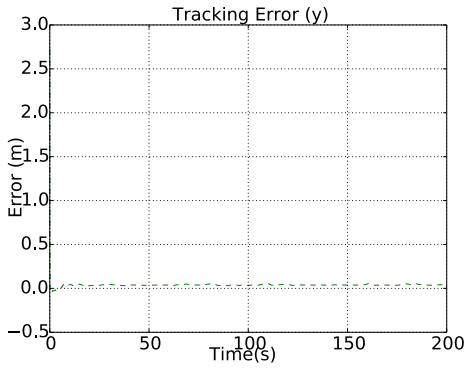
(a) Tracking Error in x of the red robot



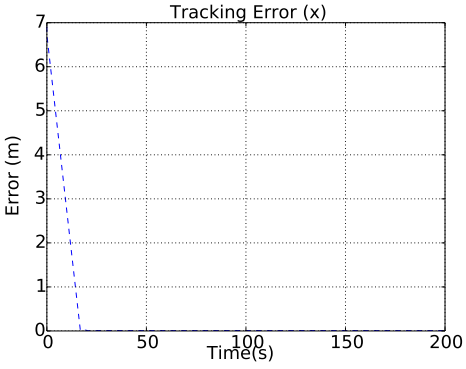
(b) Tracking Error in y of the red robot.



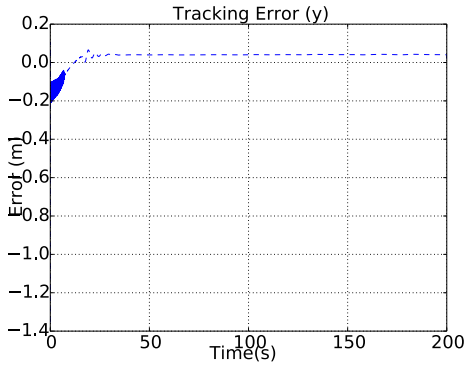
(c) Tracking Error in x of the green robot.



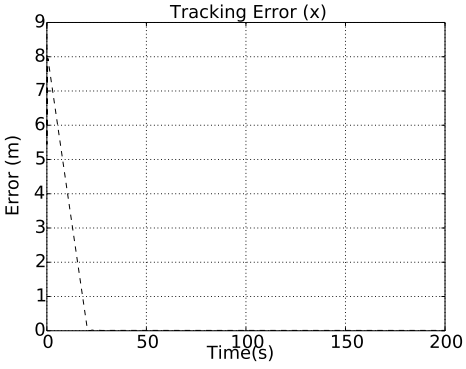
(d) Tracking Error in y of the green robot



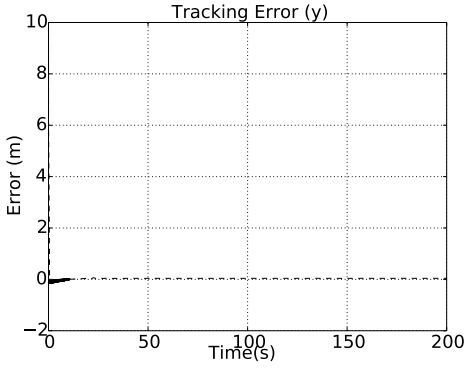
(e) Tracking Error in x of the blue robot.



(f) Tracking Error in y of the blue robot.



(g) Tracking Error in x of the black robot.



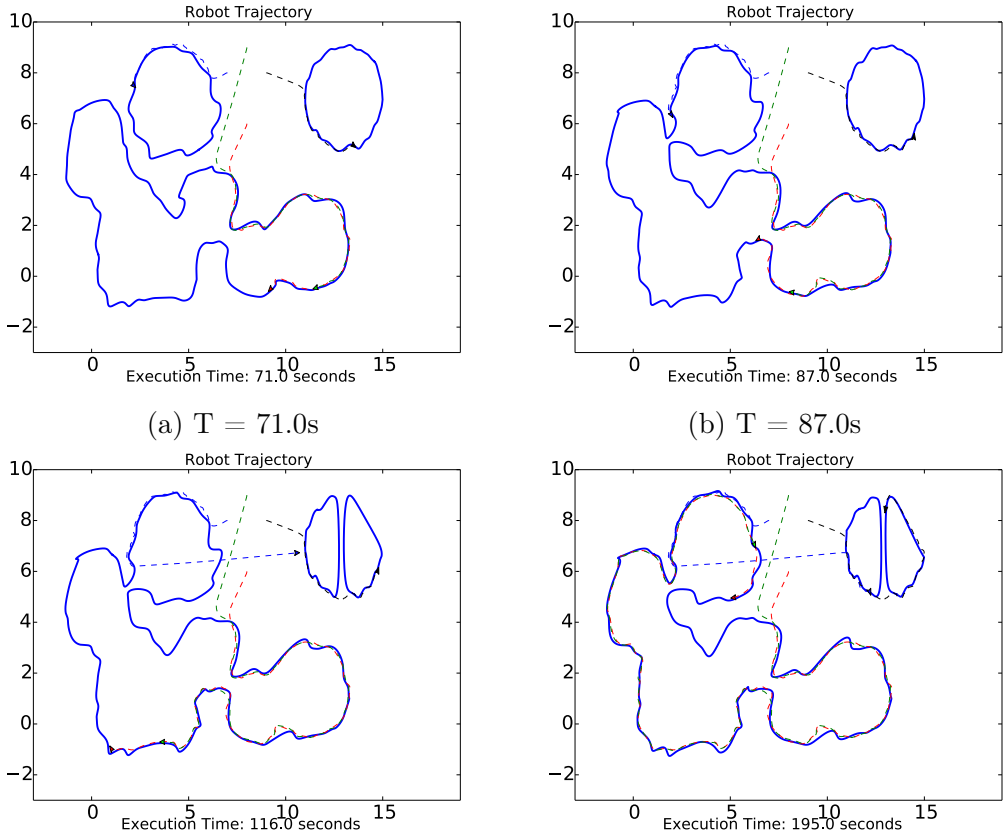
(h) Tracking Error in y of the black robot.

Figure 4.20: Experiment 6: Tracking errors in x and y for all vehicles.

4.6 Multiple Dynamic Perimeters Tracking

In this experiment, we use a scenario that is similar to the one of the previous experiment. This time, we added a dynamic behavior to the samples that represent the perimeters, so the possibilities of a perimeter merge and/or split exists. In this experiment, two perimeters close to each other are susceptible to merge, whilst the other perimeter, which is a bit isolated from the others will expand until it splits in two.

Once again, our vehicles start as a group located on the top of the scenario. At first, the Green and the Red vehicles are assigned to patrol the largest perimeter whilst the Blue and Black vehicles are assigned to the other two ones. Simultaneously to the merge of two perimeters, the remaining one splits. Figure 4.21 shows the result of this experiment:



(a) T = 71.0s (b) T = 87.0s

(c) T = 116.0s (d) T = 195.0s

Figure 4.21: Experiment 7: Multiple dynamic perimeter tracking

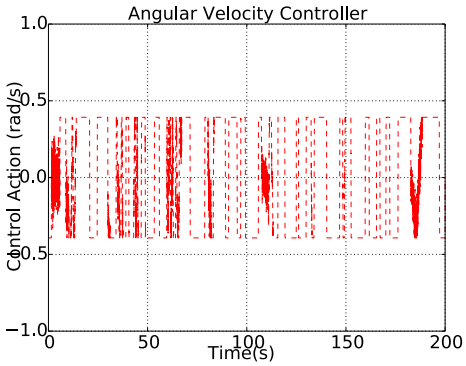
As shown in this experiment, the two expected situations – a merge and a split – occur. It is important to notice that when the split happens we end up in a scenario of four vehicles and three perimeters, in which one of the perimeters is not being covered. Due to that fact, it is necessary to reallocate one vehicle to the new area. In this case,

the Blue vehicle is the one reassigned to cover one of the perimeters. Again we see the same problem as the one in Experiment 6, in which the Blue presents the undesired behavior of passing through one of the perimeters. Although this behavior is undesired, it is not on the scope of this work to treat situations like that.

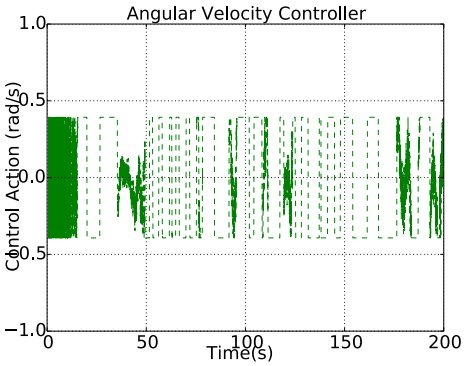
As the Blue vehicle reaches its new objective, the Black vehicle keeps the mission of tracking one of the perimeters that resulted from the split, while the Red and Green vehicle keeps on tracking the region that merged to their previously assigned perimeter, and all perimeters are covered by at least one vehicle.

It is important to notice that this experiment covers all possible situations that were described in this work: all perimeters have a random and dynamic behavior, with their samples moving in a way that it is not possible to identify a pattern or describe a dynamic system that can predict its behavior; a merge situation occurs when the boundaries of two perimeters intersect each other, resulting on the merge of two perimeters into a single one; and finally, a split situation occurs, when a group of samples of a perimeter becomes distant another group of samples of the same perimeter in a way that it is possible to create two different perimeters. Finally, it is also important to mention that this experiment

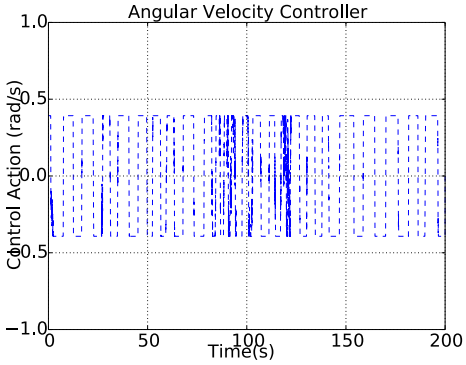
With the scenario described and the results of the experiment presented in Figure 4.21, it is important to evaluate the value of the control action on the actuators of all robots and the also the tracking error for each of them. The control action on the angular and linear velocity controllers for each of the vehicles is shown in Figure 4.22 whilst the tracking errors in x and y also for each vehicle are depicted on Figure 4.23.



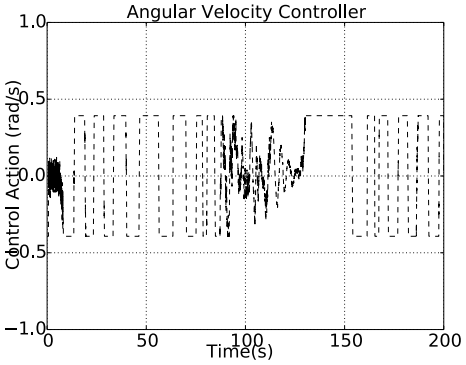
(a) Angular Velocity actuator of the red robot.



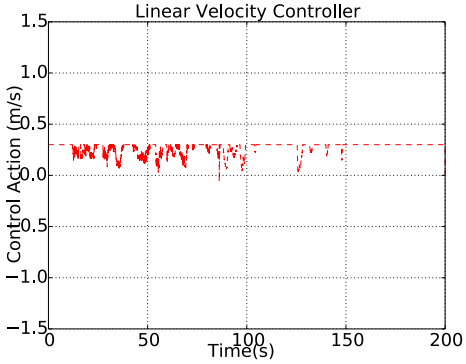
(b) Angular Velocity actuator of the green robot.



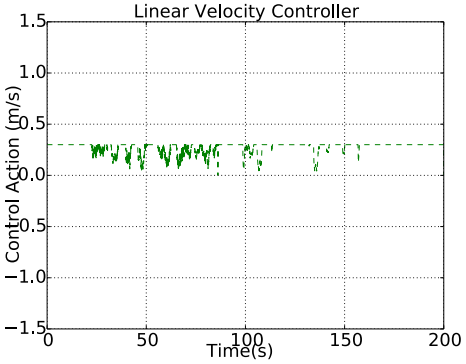
(c) Angular Velocity actuator of the blue robot.



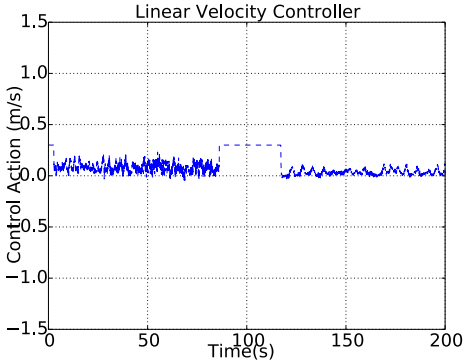
(d) Angular Velocity actuator of the black robot.



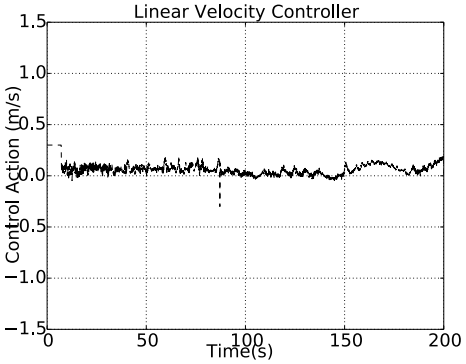
(e) Linear Velocity actuator of the red robot.



(f) Linear Velocity actuator of the green robot.

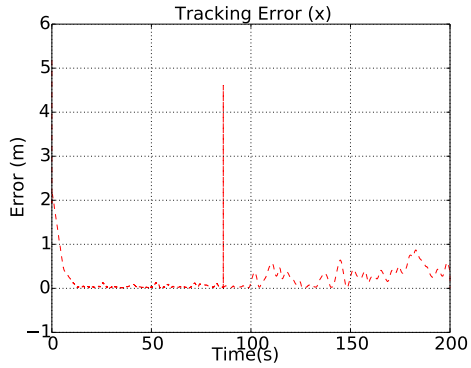


(g) Linear Velocity actuator of the blue robot.

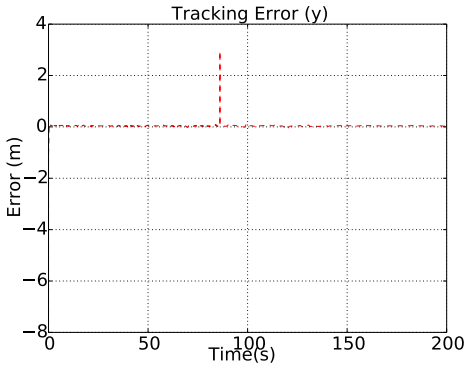


(h) Linear Velocity actuator of the black robot.

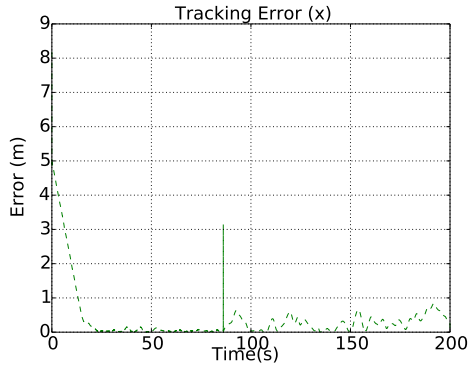
Figure 4.22: Experiment 7: Control action values for the angular velocity actuator and linear velocity actuator for all vehicles.



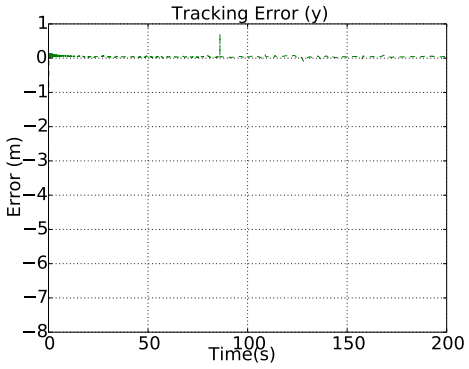
(a) Tracking Error in x of the red robot



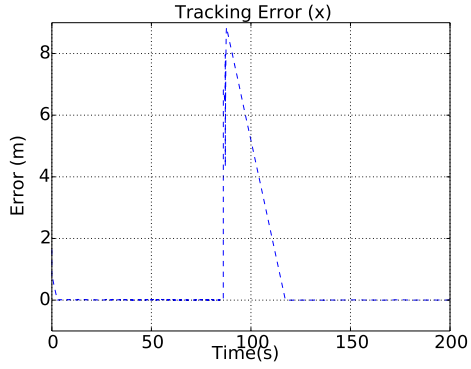
(b) Tracking Error in y of the red robot.



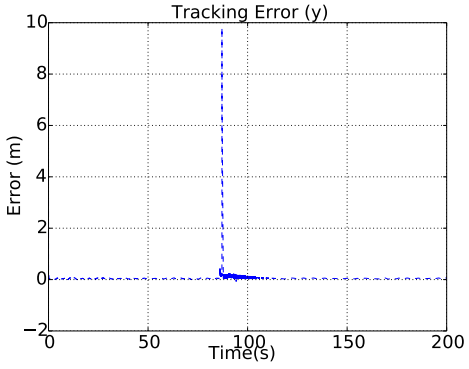
(c) Tracking Error in x of the green robot.



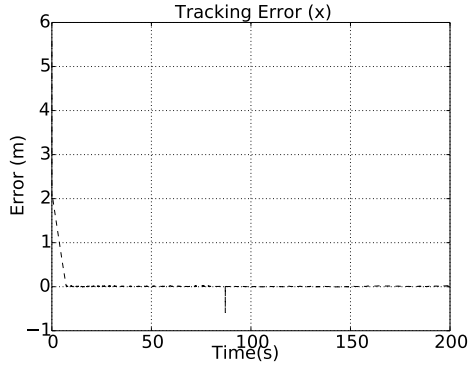
(d) Tracking Error in y of the green robot



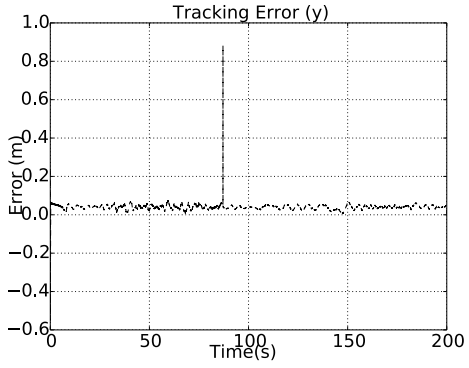
(e) Tracking Error in x of the blue robot.



(f) Tracking Error in y of the blue robot.



(g) Tracking Error in x of the black robot.

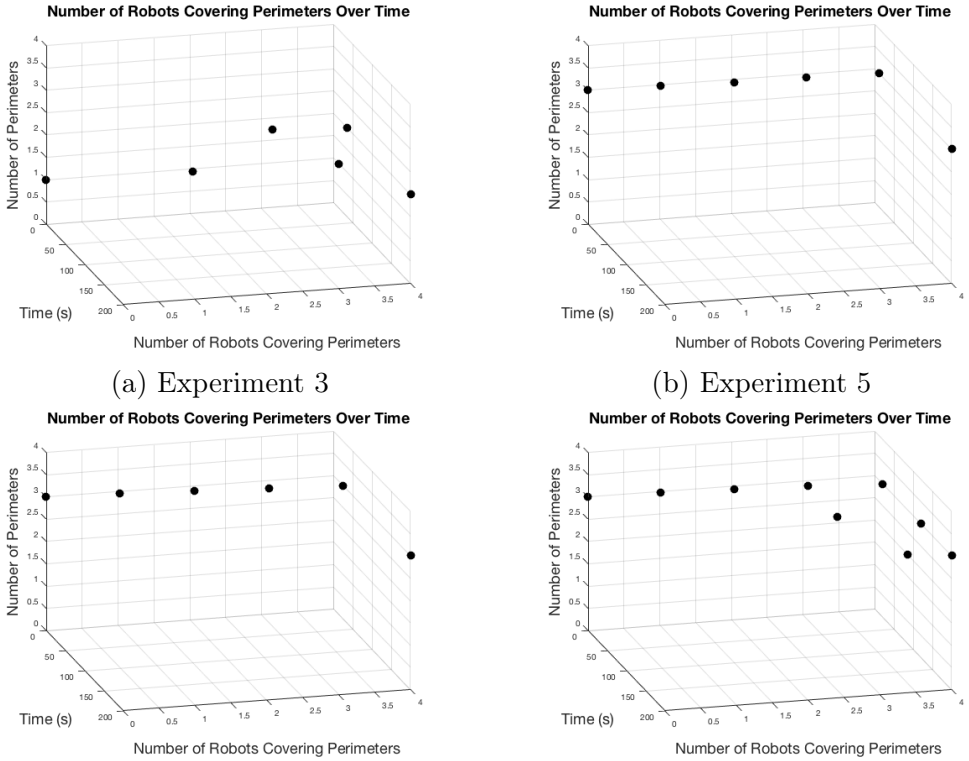


(h) Tracking Error in y of the black robot.

Figure 4.23: Experiment 7: Tracking errors in x and y for all vehicles.

4.7 Coverage Efficiency on Multiple Perimeters

On this last experiment, we revisit Experiments 3, 5, 6 and 7 to briefly address the impact of the Coverage Efficiency on the allocation strategy proposed in this work. In order to do that, Figure 4.24 shows a 3D plot of the number of allocated vehicles that are covering a perimeter over time for each one of the mentioned experiments.



(a) Experiment 3

(b) Experiment 5

(c) Experiment 6

(d) Experiment 7

Figure 4.24: Coverage Efficiency Experiment - Allocation over time given the number of vehicles and perimeters.

In Figure 4.24a, related to the split experiment, we can see that the number of perimeters is one until approximately second 26.0, when the split occurs. Right after the split, and as shown on the Figure, we can notice that now we have two perimeters and that the number of vehicles covering perimeters indicate that there is one vehicle that is not covering any perimeter. That is exactly what happens to the black vehicle, that was part of the perimeter that split and right after the split it needs to reach the curve again, therefore counting as a vehicle that was not covering any perimeter for a while. It also can be seem that the vehicle reaches the intended curve at approximately second 30.0, and from them on all vehicles are monitoring all perimeters. This shows that even after a split scenario, the black vehicle fast recalculates its *Coverage Efficiency* and goes to a perimeter in order to perform a surveillance task.

Figures 4.24b and 4.24c shows the behavior during the scenario of static curves. On both cases it is possible to see that as time goes by the number of vehicles that are covering a perimeter just increases, until all vehicles are assigned to at least one perimeter and that situation remains until the end of the simulation. As the perimeters are not dynamic, hence no split or merge occur, this is the expected behavior, and it also shows that the *Coverage Efficiency* technique tends to maintain the maximum number of areas covered, as the number of vehicles covering an area just increase until all areas are covered.

Finally, Figure 4.24d shows the last scenario with Multiple Dynamic Perimeters. As can be seen on the figure, the number of vehicles covering different perimeters just increases until it stabilizes at time around 20.0 seconds, when all areas are covered. Then, at approximately second 80.0, it is possible to notice that the number of perimeters decrease as a merge happens, but the number of vehicles covering remains the same. Then, at approximately second 90.0 when a split occurs, it is possible to notice that the number of perimeters increase and the number of vehicles covering regions also decreases. This happens because the blue vehicle is then reassigned to cover one of the perimeters that is part of the split and that was previously uncovered. Until the blue vehicle reaches its goal, at approximately second 120.0, we have a situation of four robots and three perimeters but one vehicle (the blue one) not actively tracking a perimeter. As soon as the blue vehicle reaches its goal all perimeters are covered and this situation remains until the end of the simulation. The *Coverage Efficiency* can be seem in this scenario in two situations: the first one, when the perimeters merge, the blue vehicle recalculates its coverage efficiency and as it is too close to the perimeter that it is already tracking it stays tracking it. But when the split occurs, and a new perimeter with no vehicles allocated to it emerges, the blue recalculation of the *Coverage Efficiency* of the blue vehicle leads it to the new perimeter. Once again, we demonstrate that the *Coverage Efficiency* metric tends to maintain all perimeters covered.

Chapter 5

Conclusions and Future Work

In this chapter, we discuss the conclusions that can be taken from the experiments presented in Chapter 4, and what we believe are future works that can be studied using ours as a preliminary work.

5.1 Conclusions

In this work, we addressed the problem of multiple dynamic perimeters being tracked by a group of robots. It is possible to summarize that the problem addressed in this work can be divided into three main problems that must be dealt with simultaneously:

- How to efficiently represent a perimeter: a problem that arises in this work as the samples that represent such perimeter are under constant modifications due to their dynamic behavior.
- Propose a navigation technique in which the vehicles that will be used to perform a perimeter tracking task must navigate on the boundaries of the perimeters and maintain the tracking activity as the perimeter shape changes due to its dynamic behavior.
- Allocate the vehicles to their designated perimeters so they can navigate on its boundaries and perform a surveillance task.

A solution for each of the three problems mentioned above was provided in this work: by using *Basis Spline* interpolation we managed to efficiently represent the perimeter, as for each re-interpolation the computational cost was not a problem and the generated shapes for the perimeters as a continuous polygon were extremely accurate when compared to the scattered samples. When it comes to the navigation

technique, the control strategy adopted in this work based on a feed-forward controller for a differential vehicle proved to be a good strategy once the vehicle was capable of following the perimeter to which it was assigned, even if – in certain situations and due to the limitations imposed to the values on the controllers – sometimes the vehicle did not navigate straight on the boundary but inside the perimeter. Finally, when it comes to the allocation, our proposed metric – the *Coverage Efficiency* – played an important role for us to decide how to allocate vehicles in different perimeters when it was necessary, and can be seen as a good strategy for defining utility values for different types of perimeters.

The main focus of this work – perform tracking of dynamic perimeters – was achieved. On our experiments we tested all scenarios using multiple robots: from a single perimeter that expands or shrinks, passing through an experiment in which a perimeter splits in two, and another experiment where two perimeters merge once their samples intersect each other, to experiments with multiple perimeters in static and dynamic scenarios, in which all of the previous situations – expansions, shrinks, merges and splits occur. In all those experiments we guaranteed that the limitations on the actuators were indeed followed and that the error while tracking the perimeter on both x and y axis tends to zero as the vehicle approaches and maintains itself on the boundary of the perimeter.

As described before, the contributions of this work can be seen as a methodology for dealing with dynamic perimeters that may merge or split and a navigation strategy based on feed-forward and potential fields techniques for trajectory tracking. It is important to mention that such kind of strategy was only tested – in the scope of this work – when the trajectories that we are supposed to track are represented by *Basis Splines*.

Finally, another contribution relies on the fact that by limiting the values of the actuators of the system to values that actual robots can handle, we provided a technique that can be used in actual robots that follow the model of a differential robot. Since the values on the actuators do reflect possible values on actuators of vehicles that follow such model, our simulations and strategies adopted indeed provide a good background for implementing it on actual robots.

5.2 Future Work

From the current state of this work, many possibilities for future work are possible. One that is imperative is to study how to avoid vehicles from passing inside perimeters

that are blocking their way to their objective. As shown in Experiments 6 and 7, in order to go to their objective, the vehicles ignore the existence of a perimeter on their path. In scenarios in which the exposure of the vehicle to the content of what the perimeter is limiting can harm or disable the vehicle, our navigation technique will likely fail. Therefore, adding strategies for the robots to avoid passing through the regions that are on their way to their objective to our current strategy is considerable future work.

Another possible future work that is interesting considering what was achieved in this one is to modify the vehicle's model. In this work, we use the model of a differential vehicle, but for perimeter surveillance tasks the usage of drones is most likely the first option considered. Also, depending on the task, different models of vehicles can be used. It is important to mention that our perimeter representation technique and allocation techniques do not take into account the model of the vehicle adopted, which makes it versatile and possible to be used by different kinds of vehicles.

As mentioned in the previous section, implementing our strategy as a whole on actual robots is also a future work that must be considered. Whilst our simulations showed that the adopted strategy does provide feasible results when considering the values on the actuators, and implementation on actual robots would be the empirical way to test it and unite all theory described in this work to practical results.

Finally, it is important to mention that another future work that can be produced given the background proposed in this work, is to analyze the behavior of the vehicles as the number of vehicles and curves increases. For example, scenarios using swarms and/or more curves than robots are good candidates for future work.

Bibliography

Acevedo, J. J., Arrue, B. C., Maza, I., and Ollero, A. (2013). Cooperative perimeter surveillance with a team of mobile robots under communication constraints. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5067–5072. ISSN 2153-0858.

Altché, F., Qian, X., and de La Fortelle, A. (2016). Time-optimal coordination of mobile robots along specified paths. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5020–5026. ISSN 2153-0866.

Bellotto, N. and Hu, H. (2009). Multisensor-based human detection and tracking for mobile service robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(1):167–181. ISSN 1083-4419.

Calisi, D., Farinelli, A., Iocchi, L., and Nardi, D. (2007). Multi-objective exploration and search for autonomous rescue robots. *Journal of Field Robotics*, 24(8-9):763–777.

Chaimowicz, L., Michael, N., and Kumar, V. (2005). Controlling swarms of robots using interpolated implicit functions. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2487–2492. ISSN 1050-4729.

Choset, H., Hutchinson, S., Lynch, K., Kantor, G., Burgard, W., Kavraki, L., Thrun, S., and Arkin, R. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementation*. A Bradford book. Prentice Hall of India. ISBN 9780262033275.

De Boor, C. (1986). B(asic)-spline basics. Technical report, WISCONSIN UNIV-MADISON MATHEMATICS RESEARCH CENTER.

Ding, X. C., Rahmani, A. R., and Egerstedt, M. (2010). Multi-uav convoy protection: An optimal approach to path planning and coordination. *IEEE Transactions on Robotics*, 26(2):256–268. ISSN 1552-3098.

Dong-Shu, W. and Hua-Fang, Y. (2011). Path planning of mobile robot in dynamic environments. In *2011 2nd International Conference on Intelligent Control and Information Processing*, volume 2, pages 691–696. ISSN .

Edelsbrunner, H. and Mücke, E. P. (1992). Three-dimensional alpha shapes. In *Proceedings of the 1992 Workshop on Volume Visualization*, VVS '92, pages 75--82, New York, NY, USA. ACM.

Gerkey, B. P. and Mataric, M. J. (2002). Sold!: auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758–768. ISSN 1042-296X.

Goncalves, V. M., Pimenta, L. C. A., Maia, C. A., Dutra, B. C. O., and Pereira, G. A. S. (2010). Vector Fields for Robot Navigation Along Time-Varying Curves in n-Dimensions. *IEEE Transactions on Robotics*, 26(4):647–659. ISSN 1552-3098.

Goncalves, V. M., Pimenta, L. C. A., Maia, C. A., and Pereira, G. A. S. (2009). Artificial vector fields for robot convergence and circulation of time-varying curves in n-dimensional spaces. In *2009 American Control Conference*, pages 2012–2017. ISSN 0743-1619.

Gordon, W. J. and Riesenfeld, R. F. (1974). B-spline curves and surfaces. In *Computer aided geometric design*, pages 95--126. Elsevier.

Jafari, O. H., Mitzel, D., and Leibe, B. (2014). Real-time rgb-d based people detection and tracking for mobile robots and head-worn cameras. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5636–5643. ISSN 1050-4729.

Jahn, A., Alitappeh, R. J., Saldaña, D., Pimenta, L. C. A., Santos, A. G., and Campos, M. F. M. (2017). Distributed multi-robot coordination for dynamic perimeter surveillance in uncertain environments. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 273–278. ISSN .

Jahn, A. and Pimenta, L. C. A. (2016). Sampling based path planning and vector fields for curve tracking by uavs. In *2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR)*, pages 223–228. ISSN .

Jevtic, A., Gutierrez, A., Andina, D., and Jamshidi, M. (2012). Distributed bees algorithm for task allocation in swarm of robots. *IEEE Systems Journal*, 6(2):296–304. ISSN 1932-8184.

Kim, H. and Kim, B. K. (2014). Online minimum-energy trajectory planning and control on a straight-line path for three-wheeled omnidirectional mobile robots. *IEEE Transactions on Industrial Electronics*, 61(9):4771–4779. ISSN 0278-0046.

Kim, K. B. and Kim, B. K. (2011). Minimum-time trajectory for three-wheeled omnidirectional mobile robots following a bounded-curvature path with a referenced heading profile. *IEEE Transactions on Robotics*, 27(4):800–808. ISSN 1552-3098.

Kingston, D., Beard, R. W., and Holt, R. S. (2008). Decentralized perimeter surveillance using a team of uavs. *IEEE Transactions on Robotics*, 24(6):1394–1404. ISSN 1552-3098.

Kjærgaard, M., Andersen, N. A., and Ravn, O. (2014). Generic trajectory representation and trajectory following for wheeled robots. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4073–4080. ISSN 1050-4729.

Ko, I., Kim, B., and Park, F. C. (2014). Randomized path planning on vector fields. *Int. J. Rob. Res.*, 33(13):1664--1682. ISSN 0278-3649.

Koenig, S. and Likhachev, M. (2005). Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, 21(3):354–363. ISSN 1552-3098.

Koren, Y. and Borenstein, J. (1991). Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pages 1398–1404 vol.2. ISSN .

Kroger, T., Tomiczek, A., and Wahl, F. M. (2006). Towards on-line trajectory computation. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 736–741. ISSN 2153-0858.

Lau, B., Sprunk, C., and Burgard, W. (2009). Kinodynamic motion planning for mobile robots using splines. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2427–2433. ISSN 2153-0858.

LaValle, S. M. and James J. Kuffner, J. (2001). Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400.

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., and Thrun, S. (2005). Anytime dynamic a*: An anytime, replanning algorithm. In *Proceedings of the Fifteenth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'05, pages 262--271. AAAI Press.

Likhachev, M., Gordon, G., and Thrun, S. (2003). Ara*: Formal analysis.

Michael, N., Zavlanos, M. M., Kumar, V., and Pappas, G. J. (2008). Distributed multi-robot task assignment and formation control. In *2008 IEEE International Conference on Robotics and Automation*, pages 128–133. ISSN 1050-4729.

Raja, P. and Pugazhenthi, S. (2012). On-line path planning for mobile robots in dynamic environments. *Neural Network World*, 22(1):67.

Ruangpayoongsak, N., Roth, H., and Chudoba, J. (2005). Mobile robots for search and rescue. In *IEEE International Safety, Security and Rescue Rototics, Workshop, 2005.*, pages 212–217. ISSN 2374-3247.

Saldaña, D., Alitappeh, R. J., Pimenta, L. C. A., Assunção, R., and Campos, M. F. M. (2016). Dynamic perimeter surveillance with a team of robots. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5289–5294. ISSN .

Saldaña, D., Assunção, R., and Campos, M. F. M. (2015). A distributed multi-robot approach for the detection and tracking of multiple dynamic anomalies. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1262–1267. ISSN 1050-4729.

Shiller, Z. (2015). *Off-Line and On-Line Trajectory Planning*, pages 29--62. Springer International Publishing, Cham.

Shiroma, N., Chiu, Y.-H., Sato, N., and Matsuno, F. (2005). Cooperative task excecution of a search and rescue mission by a multi-robot team. *Advanced Robotics*, 19(3):311–329.

Siegwart, R. and Nourbakhsh, I. R. (2004). *Introduction to Autonomous Mobile Robots*. Bradford Company, Scituate, MA, USA. ISBN 026219502X.

Stachniss, C. and Burgard, W. (2005). Mobile robot mapping and localization in non-static environments. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3*, AAAI'05, pages 1324--1329. AAAI Press.

Tokekar, P., Karnad, N., and Isler, V. (2014). Energy-optimal trajectory planning for car-like robots. *Autonomous Robots*, 37(3):279--300. ISSN 1573-7527.

Toraichi, K., Katagishi, K., Sekita, I., and Mori, R. (1987). Computational complexity of spline interpolation. *International Journal of Systems Science*, 18(5):945–954.

Walcott-Bryant, A., Kaess, M., Johannsson, H., and Leonard, J. J. (2012). Dynamic pose graph slam: Long-term mapping in low dynamic environments. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1871–1878. ISSN 2153-0866.

Wang, C.-C. and Thorpe, C. (2002). Simultaneous localization and mapping with detection and tracking of moving objects. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 3, pages 2918–2924 vol.3. ISSN .

Wolf, D. and Sukhatme, G. S. (2004). Online simultaneous localization and mapping in dynamic environments. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, volume 2, pages 1301–1307 Vol.2. ISSN 1050-4729.