DESIGNING A MULTI-AGENT FRAMEWORK FOR UNMANNED AERIAL/GROUND VEHICLES

WILLSON AMALRAJ AROKIASAMI

NATIONAL UNIVERSITY OF SINGAPORE

2016

DESIGNING A MULTI-AGENT FRAMEWORK FOR UNMANNED AERIAL/GROUND VEHICLES

WILLSON AMALRAJ AROKIASAMI

(*M.Sc, NUS*)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2016

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Willson Amalraj Arokiasami

December 3, 2016

Name	:	Willson Amalraj Arokiasami
Degree	:	Doctor of Philosophy
Supervisor(s)	:	Associate Professor Tan Kay Chen, Associate Professor Dipti Srinivasan
Department	:	Department of Electrical & Computer Engineering
Thesis Title	:	Designing A Multi-Agent Framework for Unmanned Aerial/-Ground Vehicles

Summary

This thesis focuses on developing a multi-agent framework that allows seamless usage of Unmanned Aerial Vehicle (UAV) or Unmanned Ground Vehicle (UGV) for the same mission. Multi-agent architectures for autonomous mobile robots are generally mission and platform oriented. Autonomous mobile robots are commonly employed in patrolling, surveillance, search and rescue and human-hazardous missions. Irrespective of the differences in unmanned aerial and ground robots, the algorithms for obstacle detection and avoidance, path planning and path-tracking can be generalized. Service Oriented Interoperable Framework for Robot Autonomy (SOIFRA) developed in this work is an interoperable multi-agent framework focusing on generalizing platform-independent algorithms for unmanned aerial and ground vehicles. SOIFRA is behaviour based and is interoperable across unmanned aerial and ground vehicles.

As obstacle detection and avoidance are standard requirements for autonomous operation of mobile robots, platform-independent collision avoidance algorithms are incorporated into SOIFRA. Obstacle detection and avoidance are performed utilizing computer vision-based algorithms, as these algorithms are generally

platform independent. Obstacle detection is achieved utilizing Hough transform, Canny contour and LucasKanade sparse optical flow algorithm. Collision avoidance is performed utilizing time-to-contact estimation techniques. In order to demonstrate the modularity of SOIFRA, two different and well known time-to-contact estimation algorithms, optical flow-based and expansion of object-based time-to-contact algorithms, are utilized for collision avoidance. Collision avoidance behaviour of SOIFRA is demonstrated through several simulation and real-time experiments that require the robot to navigate an unknown environment with obstacles. Experiments performed, utilizing Turtlebot, Clearpath Robotics Husky, AR Drone and Hector-quadrotor, establish SOIFRAs interoperability across several robotic platforms and successful incorporation of collision avoidance behaviour into SOIFRA.

Path-planning, path-generation and following a planned path successfully are fundamental requirements for autonomous operation of unmanned robots as most of these robots increasingly utilized in patrolling, surveillance, search and rescue type of missions. Though the operational principle of aerial and ground robots are different, the algorithms for path-planning and path-following can be generalized. Vector Directed Path Generation and Tracking (VDPGT) developed in this work is a platform independent real-time path-generation and path-following algorithm. VDPGT is designed to dynamically adapt the shortest path to a destination while minimizing the tracking error. Performance of VDPGT algorithm is studied through several simulation and real-time experiments that require the robot to reach several waypoints where the next waypoint is available only after reaching the current waypoint. In order to demonstrate that VDPGT is platform independent, a mission that requires UAV and UGV to cooperatively navigate a GPS denied environment using only computer vision algorithms and VDPGT is conducted. Both the robots utilize VDPGT to dynamically generate path and follow the path generated in semi as well as fully autonomous modes. In semi-autonomous mode the UAV is controlled by a human operator and UGV is fully autonomous. In fully-autonomous mode both the UAV and UGV are fully autonomous. VDPGT algorithm is incorporated into SOIFRA to provide

path-generation and path-following behaviours for unmanned aerial and ground vehicles. An AR Drone and a Turtlebot are made to reach a target destination in an unknown environment with obstacles to demonstrate the path-generation and path-following ability of SOIFRA.

With the advancements in autonomous mobile robot technologies, scenarios where human and autonomous mobile robots coexist are becoming realistic. To ensure safe human-robot coexisting environments, the autonomous mobile robots should be able to detect and avoid human collisions effectively. A biologically inspired, human visual system based human detection and human collision avoidance system is developed to add human collision avoidance behaviour into SOIFRA. Mathematical models of parvo and magno channel of human retina are combined with existing pedestrian detection algorithms such as Histogram of Oriented Gradients (HOG) and C^4 human detector (C^4) to improve pedestrian detection accuracy. The performance of the developed algorithm is analysed through comparisons with existing state-of-the-art pedestrian detectors on MIT, INRIA, Caltech and Daimler datasets. The proposed model is incorporated into SOIFRA and SOIFRA's human collision avoidance behaviour demonstrated through a series of real-time experiments. Real-time experimental results from AR Drone and Turtlebot demonstrate the improved performance, of the developed algorithm, in human detection and human collision avoidance.

The platform independent collision avoidance, path-planning and path-following, and human collision avoidance algorithms incorporated into SOIFRA increase SOIFRA's usefulness. Various simulation and real-time experiments conducted demonstrate the interoperability and modularity of SOIFRA.

Keywords : Multi-agent, collision avoidance, time-to-contact, path-generation, path-following, pedestrian detection, pedestrian collision avoidance, UAV, UGV

Acknowledgment

First and foremost, I would like to express my utmost gratitude to my parents for their support and love, which helped me cross through many tough situations with ease. Next I would like to thank my supervisors, Associate Professor Tan Kay Chen and Associate Professor Dipti Srinivasan, for their guidance and support throughout my PhD. They gave me the freedom to explore and constantly guided me. I would also like to express my deepest gratitude towards Associate Professor Prahlad Vadakkepat for his invaluable mentorship and support, which helped me to grow as a person and as a researcher.

This work could not be done without the supports, encouragements and guidance of many people. My sincere thanks also goes to my seniors Hu Jun and Yu Qiang as well as other lab buddies who helped me and shared their invaluable experience. I specially thank Gee Sen bong for being a good friend and a partner in my crimes. I would also like to thank my colleagues Lim Pin, Qiu Xin, Arrchana and Yung Siang and my juniors Zhang Chong, Sim Kuan, Gary, Ruoxu, Weinan, Rethnaraj, Sivam, Berrak, lab officers of Control & Simulation Lab, Mr. Zhang Hengwei and Ms. Sara for their continuous assistance in various tasks. All these labmates have helped me at certain point in time and helped to create a pleasant research environment.

A special thanks to Kwa Yujie, Paul Christopher who helped me with few of my experiments and other students who worked with me for their projects.

I would like to thank Arun Kumar Chandran, Josey Mathew and Subash Chandar Adikesavan for their support and patience. There are many others that I have not named who have made an impact on my life and shaped me into who I am today. My profound thanks goes to all of them. Finally, I appreciate the Department of Electrical and Computer Engineering, National University of Singapore (NUS) for providing me an opportunity to pursue PhD in Singapore.

Contents

List of Figures

List of Tables

List of Algorithms

List of Symbols

(o_x, o_y)	Optical centre of the camera
L^i	Set of n Line segments
l^i_j	j^{th} line segment obtained from image i
α^i_j	Angle associated with line segment l^i_j
θ_t	Roll angle of the robot at time t
L^i_{pr}	A set containing vertical line segments that lie to the right of the optical centre for an image i
L^i_{pl}	A set containing vertical line segments that lie to the left of the optical centre for an image i
l^i_{Or}	Vertical line segments that belongs to L^i_{pr} and lie closest to the optical centre for an image i
l^i_{Ol}	Vertical line segments that belongs to L^i_{pl} and lie closest to the optical centre for an image i
OF^i_{Or}	Optical flow vectors that are generated due to the motion of l^i_{Or}
OF^i_{Ol}	Optical flow vectors that are generated due to the motion of l^i_{Ol}
f	Focal length of the camera
t	time instant
N	A point on the obstacle
S	Image plane
n	Projection of N onto the image plane S

Z_e	Translation of robot along Z axis
V	Velocity of the robot
y_s	Distance from p to the focus of expansion for N
$\dot{y_s}$	Length of the optical flow vector
E	Expansion of an object
W	Width of the object
d_{true}	True distance to the obstacle
$\hat{d}$	Distance to the obstacle estimated
λ	Critical distance
n	Number of samples
u	control action of the LQR controller
p	current position of the robot
p^*	Position of the robot along the desired path
b	bounded region along the desired path
v_d	cross-track error velocity
q_{11}, q_{22}	Gain parameters of Q matrix in LQR controller.
(x_e, y_e, z_e)	Representation of earth coordinate frame
(x_r, y_r, z_r)	Representation of robot coordinate frame
w_s	Initial location of the robot
w_d	Destination to be reached by the robot
w_f	Final destination reached by the robot
ψ_t	Orientation of the robot in Z plane or yaw angle of the robot
ϵ_ψ	Minimum-acceptable-orientation difference
ϵ_d	Minimum acceptable error between the destination reached and destination to be reached by the robot
ψ_d	Desired orientation
$\hat{\psi_d}$	New desired orientation
ψ_d^t	Desired orientation in the current state
ψ_d^{t-1}	Desired orientation in the previous state
δ	Distance between the defined transition boundaries

tr	Minimum turning radius
$t(u, v)$	Target position for the ground vehicle in pixel coordinates
$t(x, y)$	Target position for the ground vehicle in cartesian coordinates
α	Weighing coefficient used for normalization
D_g	Diameter of the ground robot in pixels
$\chi_{(u,v)}$	Search space
W	Width of the camera sensor
H	Height of the camera sensor
pr	Photoreceptor
$C(pr)$	Adjusted luminance of the photoreceptor
$R(pr)$	Current luminance of the photoreceptor
$R_0(pr)$	Compression parameter
$L(pr)$	Local luminance
V_0	Contribution of the static compression parameter
f_s	Spatial frequency
f_t	Temporal frequency
ph	Photoreceptor network
h	Horizontal cell network
F_{ph}	Spatio temporal filter of photoreceptor network
F_h	Spatio temporal filter of Horizontal cell network
β_{ph}	Gain of spatio temporal filter F_{ph}
β_h	Gain of spatio te kkmporal filter F_h
α_{ph}	High cut-off frequency
α_h	Low cut-off frequency
τ_{ph}, τ_h	Temporal filtering constants
CgP	Local contour enhancement ability of the ganglion cells in Parvo channel
CgM	Local contour compression ability of the ganglion cells in Parvo channel
A	Amacrine cells

FgM	Low pass spatial filter
C^i_j	j^{th} contour identified from image i
A^i_j	Area of the contour j from image i
A_{thresh}	Minimum area of contour below which the contours will be discarded
$P^i{}_{(x,y)}$	Position of the pedestrian detected for an image i
$T_{P_{(x,y)}}$	Pedestrian tracklet
E_{out}	Output of Euclidean distance-based filter
$\bar{D}_{out}$	Output of direction-based filter
V_{out}	Output of velocity-based filter
Δe	Threshold for Euclidean distance-based filter
$\Delta \bar{d}$	Threshold for direction-based filter
Δv	Threshold for velocity-based filter
Str:AGT	Steering agent
Obs:AGT	Obstacle agent
PF:AGT	Path-follow agent
PG:AGT	Path-generation agent
Ped:AGT	Pedestrian agent
str:SRV	Steering service
video:SRV	Video service
TTC:SRV	Time-to-contact service
det:SRV	Obstacle detection service
pg:SRV	Path-generation service
pf:SRV	Path-follow service
pda:SRV	Pedestrian detection and avoidance service
ctr:ACT	Actions to drive the robot
pos:ACT	Actions to measure the postion of the robot
ori:ACT	Actions to measure the orientation of the robot
det:ACT	Actions to detect obstacles
track:ACT	Actions to track obstacles
TTC:ACT	Actions to compute time-to-contact the obstacles

avd:ACT	Actions to perform collision avoidance
gpath:ACT	Actions to generate path
pdet:ACT	Actions to detect pedestrians

List of Acronyms

ACTRESS	ACTor-based Robot and Equipments Synthetic System
ADP	Agent Design Patterns
AM-CW	Amplitude Modulated Continuous Wave
ARA	Autonomous Mobile Robot Architecture
ARMADiCo	Autonomous Robot Multi-agent Architecture with Distributed Coordination
AuRA	Autonomous Robot Architecture
BDI	Belief Desire Intention
BERRA	Behaviour-based Robot Research Architecture
C^4	C^4 human detector
CEBOT	Cellular Robot System
CLARAty	Coupled Layered Architecture for Robotic Autonomy
DAMN	Distributed Architecture for Mobile Navigation
DP	Detection Precision
DR	Detection Rate
FM-CW	Frequency Modulated Continuous Wave
FN	False Negative

FNR False Negative Rate

FOE Focus of Expansion

FP False Positive

FPR False Positive Rate

FPPI False Positive Per Image

GPS Global Positioning System

HDRC3 Hybrid Deliberative/Reactive

HOG Histogram of Oriented Gradients

HRI Human-Robot Interface

ICA Intelligent Control Architecture

IPL Inner Plexiform Layer

IMU Inertial Measurement Unit

LBP Local Binary Pattern

LOS Line of Sight

LQR Linear Quadratic Regulator

LTG local tangent graphs

MAE Mean Absolute Error

MAS Multi-agent System

ME Mean Error

MSE Mean Squared Error

NLGL Non-linear Guidance Law

OPL Outer Plexiform Layer

PEAS	Performance measure, Environment, Actuators and Sensors
PID	Proportional Integral Derivative
PLOS	Pure Pursuit and Line of Sight-based Path Following
PRS-Lite	Procedural Reasoning Sytem-lite
RAP	Reactive Action Packages
ROI	Region of Interest
ROS	Robot Operating System
RRT	Rapidly-Exploring Random Tree
SAR	Search and Rescue
SLAM	Simultaneous Localization and Mapping
SOIFRA	Service Oriented Interoperable Framework for Robot Autonomy
SSS	Servo, Subsumption and Symbolic
SVM	Support Vector Machines
TCA	Task Control Architecture
TDM	Target Drives Means
TOF	Time of Flight
TP	True Positive
TTC	time-to-contact
TTC-EO	Estimating time-to-contact utilizing expansion of an obstacle
TTC-OF	Estimating time-to-contact utilizing optical flow
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle

VDPGT Vector Directed Path Generation and Tracking

VF Vector Field

VFF Virtual Force Field

VOMAS Virtual Operator Multi-agent System

VTP virtual target point

Chapter 1

Introduction

"One cannot gain anything without first giving something in return. To obtain, something of equal value must be lost" – *Edward Elric*

Human society is undergoing continuous technological advancements that are leading to our increased dependency on electromechanical systems. Electromechanical systems such as robotic systems are capable of performing hazardous tasks that are difficult for humans to perform and even mundane tasks such as household chores. Unmanned robots are replacing humans in human hazardous tasks such as space and undersea exploration, remote repair and maintenance. Unmanned robots are also assisting humans in tasks such as disaster management and military reconnaissance.

Unmanned autonomous robots are machines, capable of intelligent actions and motions, operating without a guide or teleoperator. Unmanned robots have achieved significant success in industrial/manufacturing setting. Robot manipulators perform repetitive tasks such as spot welding, painting and other tasks that require speed and superhuman precision. In-spite of the success of robotic manipulators in industrial setting, they suffer from a fundamental disadvantage: lack of mobility. An industrial manipulator, always bolted down to a location, has a limited range of motion that depends on its bolted location. In contrast, a mobile robot would be free to move throughout an area, extending

its usefulness. The ability of an unmanned mobile robot to move freely in an environment, that is known or unknown, has introduced other complexities. The dynamic nature of autonomous robot's operating environment demands robots with rapid online decision-making ability, fault tolerance and scalability.

Agent oriented approaches are suitable for designing unmanned autonomous robotic systems. Agent oriented approaches break down sequential, top-down approaches into a set of simple, distributed and decentralized processes that have direct access to the sensors and actuators of the robot [79]. An agent is defined as a computer system situated in an environment, capable of flexible autonomous actions in that environment in order to meet its design objectives [58]. An agent can be considered as a component of a software or a hardware, capable of acting exactingly in order to accomplish tasks on behalf of its user [96]. Any software process that exhibits the following properties is considered as an agent [142],

1. Autonomy: the ability to operate without any direct human intervention and have control over their actions and internal state.

2. Social ability: the ability to interact with other agents and humans through any kind of agent-communication language.

3. Reactivity: the ability to perceive their environment and respond to the changes occurring in that environment in a timely fashion.

4. Pro-activeness: the ability to take initiative and exhibit a goal-directed behaviour.

The activities or missions carried out by unmanned robots are generally large-scale and complex. A single agent may not be able to provide a comprehensive solution for these realistic problems [7]. However a group or an organization of agents, in which each agent is highly specialized in solving an individual aspect of the bigger problem, may be able to provide satisfactory performance. This organization is termed as a Multi-agent System (MAS). A MAS is a loosely coupled network of agents that work together to solve problems that are beyond the capabilities or knowledge of individual agents [58]. The inherent modular and

distributed nature of a MAS offers significant advantages over other centralized control systems such as:

- Parallelism: MAS resolve problems that arise due to limited resource availability by exploiting the parallelism of the agents in the system. The risk of bottleneck and single point failure, common in centralized systems, are also avoided due to parallelism.

- System integration: Cooperating agents in MAS allows easy inter-connection and inter-operation of multiple existing legacy systems.

- Distributed control: Knowledge acquired by spatially distributed agents in a MAS provides distributed expertise that can be used to solve a problem efficiently.

In addition to the above mentioned advantages MAS offers enhanced performance of computational efficiency, reliability, extensibility, robustness, maintainability, responsiveness and re-usability. These qualities make MAS suitable for developing control architectures for unmanned robotic systems [57, 59, 74, 127, 129].

1.1 Robot Control Architectures

Multi-agent-based robot control architectures can follow deliberative, reactive or hybrid design paradigms. The deliberative or hierarchical design paradigm is the oldest, where each layer provides sub-goals to the layer below. A deliberative architecture will include a global world model which is updated through perception. Actions performed by robots are carried out based on the planning and reasoning derived from the world model. Reactive agent architectures were inspired from the idea that most of our day-to-day activities consist of routine actions rather than abstract reasoning. This resulted in an architecture that is a collection of behavioural schemes where cognition is reduced to a mapping of perceptual stimuli into primitive actions. The behavioural schemes in the architecture react to changes in the environment in a stimulus-response manner. Due to difficulties in creating a global world model and inadequate sensors used for perceiving

environments, neither purely reactive nor purely deliberative architecture perform well. Hybrid architectures are the commonly used architectures for robot control. Hybrid architectures include the high-level reasoning (to plan actions ahead in time) and low-level reactive capabilities (to respond to changes in real-time). There are many hybrid robot control architectures available. Table 1.1 shows some of the most famous architectures that were implemented and tested on real robots. The following subsections provide an overview of these architectures grouped into different design paradigms.

Table 1.1: List of some of the famous multi-agent architectures that were implemented and tested on real robots.

Year	Name
1986	Subsumption Architecture [13]
1987	Autonomous Robot Architecture - AuRA [5]
1989	ACTor-based Robot and Equipments Synthetic System Project - ACTRESS [6]
1989	Reactive Action Packages - RAP [36]
1990	Cellular Robot System - CEBOT [40]
1991	A Three-Layer Architecture for Navigating Through Intricate Situations - ATLANTIS [41]
1992	Servo, Subsumption and Symbolic Architecture - SSS [19]
1994	Task Control Architecture - TCA [121]
1997	Distributed Architecture for Mobile Navigation - DAMN [111]
1997	Autonomous Mobile Robot Architecture - ARA [93]
1997	Saphira [68]
1998	ALLIANCE [104]
2000	Behaviour-based Robot Research Architecture - BERRA [78]
2002	Yuvuz [149]
2003	Busquets [16]
2006	Tripdal Schematic Architecture [67]
2007	Virtual Operator Muti-agent System - VOMAS [50]
2008	Autonomous Robot Mult-agent Architecture with Distributed Coordination - ARMADiCo [7]
2012	A multi-robot platform for mobile robots [110]

1.1.1 Deliberative and Reactive Control Architectures

Subsumption architecture is one of the most representative architecture for reactive architectures. Subsumption architecture is a composition of hardware implementations with a number of levels of competence. The levels of competence are an informal specification of a desired class of behaviours for a robot over all

the environments it will encounter [13]. Reactive Action Packages (RAP) [36] is one of the most representative architecture for hierarchical control architectures. RAP is a three layered architecture that includes planning, execution and control components. By allowing reaction tasks to exist on the execution component concurrently, the RAP system is able to deal with situations where the planning system is not able to predict the events in the environment as well as respond immediately to new incoming information.

1.1.2 Hybrid Control Architectures

Autonomous Robot Architecture (AuRA) [5] is a schema theory based multi-agent architecture. AuRA consists of cartographic, planning, homeostatic control, perceptual and motor subsystems. Several schemas may be active at the same time in AuRA enabling the multi-agent architecture to provide multiple inputs such as steering and velocity inputs to the robot at the same time. A three layer architecture for navigating through intricate situations called ATLANTIS [41] is an asynchronous and heterogeneous architecture. ATLANTIS integrates traditional symbolic planning into real-world embedded reactive system. ATLANTIS is capable of handling multiple real time tasks in partially predictable noisy environments. Servo, Subsumption and Symbolic (SSS) architecture [19] is considered as an evolution of the subsumption architecture. SSS architecture includes servo, subsumption and symbolic systems. In SSS architecture, 40 separate processes operating in a loosely connected network of 24 processors form the robot controller. The robot controller has a number of modules, each of them forming small parts of the overall behaviour. Task Control Architecture (TCA) [121] has attributes that makes it an operating system rather than an architecture. TCA has both reactive as well as deliberative layers. TCA [111] includes a central unit that registers all the modules that are in use and handles communication between the modules. The low level tasks of TCA resemble behaviours. Distributed Architecture for Mobile Navigation (DAMN) [111] consists of several modules that concurrently share the control of the robot. The modules of DAMN architecture communicate and vote for actions that satisfy objectives and against other

actions. Behaviour-based Robot Research Architecture (BERRA) [78] is made up of deliberative, task execution and reactive layers. The deliberative layer includes a Human-Robot Interface (HRI) that understands gesture and speech inputs and a planner that decomposes data from HRI into different states. The task execution supervisor and a localizer form the task execution layer. Behaviours (coupling between sensors and actuators), resources and controllers form the reactive layer. YAVUZ [149] is a modular hierarchical architecture. YAVUZ has three operational modes: playback operation, teaching operation and manual operation. Manual operation is utilized for teleoperation while the other two modes are utilized for self-supervised and goal oriented autonomous operations. Tripodal schematic control architecture [67] is also a three layered architecture made up of deliberative, sequencing and reactive layers. Communications between components in different layers are carried out through synchronous/asynchronous and push/pull relations.

1.1.3 Single Robot Multi-agent Architectures

Autonomous Mobile Robot Architecture (ARA) [93] is a multi-agent architecture for single robots. The functionalities of robots using ARA is determined by the emergence of behaviours due to the coexistence and cooperation among the society of agents in ARA. The agents in ARA are organized into reflexive, reactive and cognitive categories. The agents in reflexive category take care of the low level tasks. The agents in reactive category are in-charge of the abilities of the mobile robot, while the agents in cognitive category decide the best actions to perform based on the knowledge about the environment. Saphira [68] architecture consists of a deliberative and reactive layer. Planning agent of the deliberative layer forms the core of the Saphira architecture. Planning agent is based on Procedural Reasoning Sytem-lite (PRS-Lite) reactive planner, a system capable of taking natural language voice commands and transforming them into tasks. PRS-Lite is represented as parametrized state machines (activity schemas). Each schema embodies procedural knowledge to attain an objective through a sequence of sub-goals, perceptual checks, primitive actions and behaviours.

Virtual Operator Multi-agent System (VOMAS) [50] is a multi-agent architecture capable of dynamic task switching in order to support spontaneous generation of a task without preplanning. VOMAS has been tested on a wheelchair robot and also for formation control. Autonomous Robot Multi-agent Architecture with Distributed Coordination (ARMADiCo) [7] is a mobile robot control architecture with distributed coordination. Introduction of new hardware and software components may be done easily without any modifications to the existing agents. In addition to this, the agents in ARMADiCo have the flexibility to be coded in different programming languages and are capable of operating in different computers.

1.1.4 Multi robot Multi-agent Architectures

ACTor-based Robot and Equipments Synthetic System (ACTRESS) [6] is a heterogeneous multi-robot architecture for performing tasks that cannot be accomplished by a single robot. Negotiation mechanisms based on Contract Net and multistage negotiation protocols were introduced to perform collaborative tasks. Cellular Robot System (CEBOT) [40] is a collection of heterogeneous robotic agents that are capable of assembling and disassembling by themselves. The ability to allow complex structures to be constructed on-site and the additional capability of reconfiguring the combined units is of great value for wide range of applications in space-constrained environments. ALLIANCE [104] architecture is a fault-tolerant, adaptive, distributed behaviour-based multi-robot control system. ALLIANCE architecture can be used for missions in dynamic environments and for complex missions composed of independent tasks. L-ALLIANCE [104] is an extension of ALLIANCE architecture. L-ALLIANCE architecture uses reinforcement learning to adjust the parameters controlling the robot's behaviours. Both ALLIANCE and L-ALLIANCE architectures have been implemented on robots for performing tasks such as box-pushing, puck-gathering and marching in formation. A multi-robot platform for mobile robots [110] deals with the foundations and methods of a middle layer that joins a multi-agent system with a multi-robot system in a generic manner. It provides a generic software platform

integrating multi-agent technology and a multi-robot system.

Unmanned robots use robot control architectures to accomplish a task or mission. Most of the times the tasks performed by unmanned robots vary depending upon the content of the mission. But generally, the common robot control architectures are not flexible to accommodate the variations in tasks. As there are many types of unmanned robots available, the same mission can be completed by utilizing different robots. Existing multi-agent /multi-robot control architectures does not provide this functionality even though the algorithms for the tasks performed by the robots can be generalized. This led to the development of a separate class of multi-agent architectures that focus on introducing flexibility and generalization among robotic systems. Table. 1.2 lists some of the well known multi-agent architectures that focus on the above mentioned functionalities.

Table 1.2: List of some of the multi-agent architectures that provide flexibility and generalization among robotic systems.

Year	Name
2003	Coupled Layered Architecture for Robotic Autonomy - CLARAty [92]
2007	Virtual Operator Multi-agent System - VOMAS [50]
2008	Agent Design Patterns [7]
2011	Target-Drives-Means - TDM [12]
2015	Hybrid deliberative/reactive architecture - HDRC3 [26]
2015	Intelligent Control Architecture [55]

Coupled Layered Architecture for Robotic Autonomy (CLARAty) is a framework for heterogeneous robot platforms with generic and reusable robotic components [92]. One of the major goal of CLARAty is to provide a design that allows usage of components, that are spanning across domains, to be flexible and extensible to support various applications. CLARAty tries to provide a framework for algorithms developed for robotic systems that can be generalized, while maintaining the ability to easily integrate platform specific algorithms. CLARAty provides a framework for generalized algorithms applied to rover platforms irrespective of the implementation details.

VOMAS is a mobile robot architecture designed to rapidly change missions through dynamic task switching or dynamic role switching. Generally in multi-

robot missions involving more than one robots, each robot will be assigned a particular task to complete. If a robot is not able to perform a specific task and if there are other robots that may be equipped with skills for that task, VOMAS architecture allows the robots to perform a task switch. The mobile agent considered in VOMAS architecture is a mobile software agent, a software program, that can migrate from host to host through a network.

Agent Design Patterns (ADP) defining common features allow introduction of new hardware and software components without modifying the architecture [7]. ADP is used as a methodological approach for designing agents. ADP simplifies the task of creating new agents by defining common features of all the agents in the architecture. This pattern helps by designing agent's goals, maintaining agent's internal states and resource allocation.

Target Drives Means (TDM) is a behaviour-based interoperable software framework for humanoid robots that is ready to use and allows easy adaptation for new projects [12]. TDM focuses more on modularizing software development for adding complex behaviors to humanoid robots. TDM provides a solution to this in the form of a supplementary light software framework on top of an existing module-based architecture. This reduces the level of technical knowledge required for delivering a complex behaviour control implementations for humanoid platforms.

Hybrid Deliberative/Reactive (HDRC3) Architecture is a distributed architecture for unmanned aircraft systems [26]. In HDRC3, the essential generic functionalities of an UAV are isolated for effective integration of low-level (navigational subsystem, low-level control with motion planning) and high-level (mission planning and execution) functionalities. HDRC3 has been tested on multi-rotor platforms for several complex missions that require high autonomy.

Intelligent Control Architecture (ICA) [55], is a generic capability-oriented architecture for autonomous marine robots. ICA enables multiple collaborating marine vehicles to autonomously carry out underwater intervention missions. ICA moves away from fixed mission plans and elementary diagnostics schemes to a more robust architecture to deal with complex underwater intervention missions.

Service oriented computing forms the foundation for ICA. In addition to service oriented computing, ICA includes a knowledge based database that captures the domain specific skills of the human expert as well as the dynamic information related to the platform and changes in the environment. The combination of the above two components enable the robots utilizing ICA to dynamically reconfigure and adapt itself in order to deal with changes in the operating environment.

1.2 Aim and Scope of the Thesis

Although there are research works that focus on creating multi-agent architectures that provide flexibility and generalization among robotic systems, there are still some open areas for exploration. Interoperability of multi-agent architectures that allows a multi-agent architecture to utilize different types of unmanned robotic systems has received less attention. CLARAty, HDRC3 and ICA are some of the multi-agent architectures that focus on interoperability across unmanned robots. These architectures focus on achieving interoperability across same type of robots. CLARAty focuses on interoperability across ground robots, HDRC3 is for unmanned aircraft systems and ICA targets autonomous marine robots. However, the rapid progress in technology and increasing complexity of tasks require multi-agent architectures that accommodates different types of robotic systems such as UAVs and UGVs with minimal modifications. The main focus of this thesis is to develop a multi-agent-based framework that achieves interoperability across varied robotic platforms.

A number of algorithms developed for robotic systems can be generalized. The algorithms for robotic systems can be utilized irrespective of their implementations on various robotic platforms. Generalizing platform independent robotic algorithms will improve the ease-of-use of MAS especially in missions that involve different types of robots. This thesis also focuses on providing a framework for platform independent robotic algorithms that is interoperable across UAVs and UGVs. Following are the two key focuses of this thesis.

- Creating a multi-agent framework that allows usage of different types of

robotic systems with minimal modifications. This has significant advantages as it reduces the need for creating new multi-agent architectures for completing same missions with different types of robotic systems.

- Providing a framework to utilize platform independent algorithms on UAVs and UGVs.

This thesis aims to develop a generalized, multi-agent-based framework that allows easy usage of UGV or UAV for same missions. The developed framework should allow easy usage of either UAV or UGV, for various missions, with minimal changes to the framework. The framework should also accommodate platform independent algorithms for performing same tasks such as collision avoidance, path-following and, pedestrian detection and avoidance using UAVs or UGVs.

1.3 Literature Review

Tasks such as collision avoidance, path-generation, path-planning, path-tracking and pedestrian detection and avoidance are essential for autonomous operation of unmanned aerial and ground vehicles. This section explores the existing works related to collision avoidance, path-planning and path-tracking, and pedestrian detection.

1.3.1 Literature Review on Existing Collision Avoidance Techniques

Collision avoidance is a crucial functionality in autonomous mobile robot navigation. Collision avoidance comprises of obstacle detection and avoidance in a three dimensional environment [140]. Although simple solutions such as probes and light beams are sufficient in recognizing obstacles in industrial environments, mobile robots rely on techniques with higher sensing rate, longer range and high spatial resolution [47]. Active ranging sensors are some of the popular sensors utilized for obstacle detection. Laser range sensors have been in active use since 1976 [94]. Most of the active ranging sensors are of low cost and provide simple

outputs such as direct measurements of distance from the robot to the objects nearby. This section provides an overview about the commonly utilized existing collision avoidance techniques.

1.3.1.1 Time-of-Flight Active Ranging

Amplitude Modulated Continuous Wave (AM-CW) laser-based, Frequency Modulated Continuous Wave (FM-CW) laser-based and Time of Flight (TOF) laser-based techniques are the three basic active laser ranging techniques. While AM-CW lasers utilize the phase shift between emitted and received laser beam, FM-CW utilize frequency shift of a frequency modulated laser for estimating range. TOF lasers utilize the propagation speed of sound or an electromagnetic wave. Though FM-CW laser-based techniques are more accurate of the three, their design is complex. AM-CW laser-based techniques, more sensitive to ambient light, are faster and perform better in indoor environments. TOF laser-based techniques with their capacity for long range data acquisition, are more suitable for mobile robot applications. A SICK laser scanner is a TOF ranging laser with a resolution of 0.5 degree with a range of 50m [80]. A major disadvantage in using laser sensors is their sensitivity to environmental conditions as lasers tend to be scattered due to fog or dust. The laser in TOF ranging laser can be replaced with an ultrasonic sensor. The ultrasonic sensor transmits a packet of ultrasonic pressure waves and measures the time taken by that wave packet to reflect and return to the receiver [120]. Ultrasonic senors are usually utilized for small ranges of 0.1 – 5m. Ultrasonic sensors are always used in arrays as a single ultrasonic sensor has a slow cycle time. The accuracy of ultrasonic sensors depend on the successful perpendicular reflection of the sound wave off the obstacle. Measurements are error prone if the angle of incidence of the ultrasonic waves are not perpendicular to the object. The material of the obstacle also plays an important role in range sensing using an ultrasonic sensor. For example, materials such as foam, fur and cloth can acoustically absorb the sound waves.

1.3.1.2 Active Triangulation

Active triangulation is one of the initial approaches for range sensing in mobile robots. Sensors based on active triangulation use geometric properties manifest in their measuring strategy to estimate the distance to objects [120]. The active triangulation-based sensors project a known light pattern onto the environment and the reflection of the known pattern is captured by a receiver. By combining the information from the captured reflection pattern and known geometric values, simple triangulation techniques are utilized to establish range measurements. The sensor can be 1D or 2D based on the measurements captured by the receiver. If the receiver measures the position of the reflection along a single axis, the optical triangulation sensor is 1D and if the receiver measures the position of the reflection along two orthogonal axes then the optical triangulation sensor is considered 2D (structured light sensor). The resolution of the triangulation system is directly related to the baseline (separation between the laser and the optical centre of the camera). When the baseline increases, resolution relative to depth of field increases as well as occurrences of occlusions and missing data. This can be avoided by scanning both the emitted beam and the receiver optics [109]. This approach enables the projected beam to remain in focus over a large depth of field maintaining high resolution in-spite of a short physical baseline. The frame rate of the scanners also play a role in active triangulation as images must be acquired and processed for each position of the beam. Projecting structured pattern over the entire scene and estimating depth through triangulation by processing only few images helps in overcoming the frame-rate data acquisition problem. Although active triangulation provides an inexpensive alternative to the laser range-finding solutions, increasing quality of the laser range finding systems in the recent years have made active triangulation techniques less popular [120].

1.3.1.3 Motion or Speed-based Range Sensors

Motion or speed-based range sensors directly measure the relative motion between the robot and its environment [120]. These sensors will be be able to detect

relative motion and estimate speed as long the object moves relative to robot's reference frame. Doppler effect-based sensor is one of the common motion-based range sensor. If an electromagnetic or sound wave is transmitted and received after reflection from an object, the measured frequency at the receiver is relative to the speed between transmitter and receiver. Both microwaves and laser radar systems are utilized for Doppler effect-based range sensing in autonomous robots as well as manned trucks [141]. Both the systems have equivalent range, but performance of laser-based sensors is reduced in environments with fog, rain and dust.

1.3.1.4 Vision-based Sensors

Vision based obstacle detection is powerful and popular in unmanned aerial and ground vehicles [70, 154]. Vision sensors, in close proximity, provide detailed information about an environment. Appearance-based and optical-flow-based techniques are commonly used in map-less vision-based navigation [89]. Appearance-based methods rely on basic image processing techniques to differentiate the obstacles from the background. Optical-flow-based techniques utilize apparent motion of objects in scenes [89]. Optical flow, a biologically inspired approach for obstacle avoidance, is the pattern of apparent motion of pixels in successive image frames [128]. The time derivative of positions of image points in an image plane define a motion-field. Temporal change in an image sequence constitutes the optical flow field which approximates the motion field. Optical flow is computed using spatio-temporal intensity derivatives (differential method), feature matching techniques (correlation approach) and velocity tuned filters (frequency-based method) [154]. Lucas-Kanade method, a differential method for optical flow estimation, is useful for sparse optical flow computation [81]. For larger pixel motions, a pyramidal approach for Lucas-Kanade method is suitable, where the standard algorithm is applied recursively to re-sized versions of the image.

1.3.1.5 Obstacle Avoidance

Obstacle avoidance algorithms generate motion instructions by combining distance to the obstacle(s) and vehicle motion information. Range sensors are often utilized to obtain distance information from environment. Recent advancements have made computer vision algorithms capable of estimating depth information utilizing multiple image frames. Obstacle avoidance algorithms are of two categories: global and local. Global approaches compute optimal robot trajectory offline, utilizing a complete model of robot's environment while computationally efficient local or reactive approaches generate sub-optimal robot trajectories. Local approaches such as bug algorithm, potential field method (PFM), virtual force field (VFF), vector field histogram (VFH) and dynamic window approach (DWA) are fast and computationally efficient as only a small subset of obstacles close to the robot is considered. Bug algorithm is one of the simplest solution for obstacle avoidance. There are two variants of bug algorithm: bug1 and bug2. In bug1, the robot fully circles an obstacle detected and departs from the point with the shortest distance to the goal [83]. Though this approach is inefficient, it guarantees that the robot will reach any reachable goal. In bug2, the robot begins to follow the obstacle's contour, but departs immediately once the robot is able to move towards its goal [82]. Tangent Bug is a variant of bug algorithm that allows the robot to move efficiently using local tangent graphs (LTG) [63]. In potential field method [66], a robot is subjected to attractive force from the target and repulsive forces from obstacles. Virtual force field and vector field histogram are extensions of PFM [134]. In the dynamic window approach, robot dynamics is considered to compute admissible velocities for safe robot motions [38]. Stereo vision is a common technique for obtaining 3D depth information from 2D planar images [2].

1.3.2 Survey on Path-planning Techniques for Mobile Robots

Path-planning algorithms are generally classified into two categories: global path-planning and local path-planning algorithms. Global path-planning algorithms

require complete information about the robot's operating environment. The robot's operating environment may be represented as a continuous geometric description, decomposition-based geometric map or a topological map. A path-planning system transforms this continuous environmental model into a discrete map suitable for a path-planning algorithm. Path-planning algorithms are grouped into different sub-categories based on the manner in which this discrete decomposition is carried out.

1.3.2.1 Roadmap Approaches for Path-planning

Roadmap approaches capture the connectivity of the robot's free space in a network of 1D curves or lines called roadmaps [120]. Roadmap approaches to path-planning reduce the problem to that of a graph search by fitting a graph to the space [44]. The main challenge for roadmap-based approaches would be to enable the robot to reach any location in the operating environment by utilizing a minimum number of total roads. Following are some of the common roadmap-based approaches for path-planning.

Visibility Graph

Visibility graph path-planning is useful in cases where objects in the environment are represented as polygons in either continuous or discrete space. This approach builds roadmap of lines by connecting each vertex with all vertices visible from its position. The straight lines connecting these vertices form the shortest path between them. The performance of this method is fast when the environment is sparse and it becomes slow and inefficient, compared to other techniques, in densely populated environments. In addition to this the minimum-length path tend to take the robot as close as possible to the obstacles in the robot's path. Alternative to this approach is Edge-sampled visibility graph that assigns multiple vertices along edges of polyhedral approaches so that there is a minimum edge length.

Voronoi Roadmap

Voronoi roadmap approaches tend to maximize the distance between the robot and the obstacles in the environment. Voronoi roadmap approaches build edges

that are maximally distant from the obstacles and find the minimum distance path along these edges. Since voronoi roadmap approaches maximize the distance between the robot and the obstacles, planned paths for robots with short-range sensors will be inefficient.

1.3.2.2 Path-planning Approaches based on Cell Decomposition

Cell decomposition techniques distinguish between geometric areas or cells that are free and the areas that are occupied by objects. The operational mechanism for cell decomposition path-planning is as follows [112]:

- Divide the operating region into simple connected regions. These divisions are termed as cells.
- Identify the cells that are free of obstacles and free cells that are adjacent to each other. These open cells that are adjacent are then connected utilizing graphs called connectivity graphs.
- Locate the cells that contain the initial and goal states and search for a path in the connectivity graph that joins the initial and goal cell.
- Compute a path within each cell that connects the initial and final goal states.

Placement of the boundaries between the cells is an important aspect for cell decomposition methods. Based on the placement of the boundaries between the cells, cell decomposition methods are classified as exact or approximate cell decomposition techniques. In exact cell decomposition techniques, the boundaries of the cells are placed as a function of the structure of the environment so that the cell decomposition is loss-less. In approximate cell decomposition the decomposition results in an approximation of the actual map.

Exact Cell Decomposition

In exact cell decomposition, the boundaries of the cells are decided based on the geometric structure of the objects in the robot's operating environment. The decomposed cells are either completely free or completely occupied. The free

cells are then connected by a graph and searched using a graph search. There are different approaches of exact cell decomposition based on the manner in which the free cells are divided. Trapezoidal decomposition divides free cells into trapezoidal regions [148]. Vertical lines from each of the obstacles are used to divide the free space. A roadmap is then constructed by connecting the midpoints of trapezoids nearby. Critical-curve-based decomposition divides free space into critical and non-critical regions [116]. The boundaries of these regions are piecewise polynomial curves. Cylindrical algebraic decomposition extends critical-curve decomposition to three dimensional problems. Connected balls in free space is an exact cell decomposition approach that deals with un-structured obstacle fields [135]. This approach fills free space in the operating region with overlapping balls. The main disadvantage of exact cell decomposition approaches is that the total number of cells depend on the density and the complexity of the objects in the environment. This has a direct effect on the computational efficiency of the path-planning algorithm.

Approximate Cell Decomposition

Approximate cell decomposition is similar to grid-based environmental representations. Rectanguloid cell decomposition divides the entire operating region into rectanguloid regions [17]. Each rectanguloid are then labeled as completely filled, partially filled or completely empty. A^* and D^* algorithms that search over a square or cubic grid of unoccupied cells are the most common examples of rectangulaoid cell decomposition. Octtree decomposition is designed to reduce the number of points needed to represent obstacles, compared to full grid representations [34]. Approximate-and-decompose decomposition approach is similar to trapezoidal decomposition. The difference is that the trapezoidal regions are replaced with rectangular mixed regions to reduce the proportion of mixed area [44].

1.3.2.3 Potential Field Path-planning

Potential field methods for path-planning assign potential function to free space and consider the robot as a particle that reacts to the forces due to the potential

field [54]. The potential field method treats the robot as a point under the influence of an artificial potential field. The robot moves by following the field. The goal position has the lowest potential and attracts the robot towards it, while the obstacles repel the robots. The superposition of all forces applied on the robot guides the robot towards its goal avoiding the known obstacles in its path. Potential field is to be updated if new obstacle appear in the environment. Virtual Force Field (VFF) is the oldest potential field method [29]. The goal point is assigned a decaying function with negative minimum value and the obstacles are assigned a decaying function with positive maximum value. The forces from the goal and all the obstacles are added to obtain the total potential. VFF method suffers from local minima. Arbitrary potential field method is designed for potential fields that suffer from local minima. This method replaces gradient descent with a search that is complete in probabilistic sense. Harmonic potential functions utilize partial differential equations such as Laplace's equation, Poisson's equation, conduction heat flow equation and approximations to Navier-Stoke's equation for generating paths [1, 64, 150]. These equations generate paths that are smooth and have only one local minima at the goal. In extended potential field method two additional potential field other than the artificial potential field are utilized [65]. Rotation potential field assumes that the repulsive force is a function of the distance from the obstacle and orientation of the robot relative to the obstacle. This improved the wall following ability of the robot (difficult using standard potential field methods). The task potential field filters obstacles that should not affect the near-term potential using the current robot velocity as reference. This improved the smoothness of the generated trajectories.

1.3.2.4 Rapidly-Exploring Random Tree (RRT)

Rapidly-Exploring Random Tree (RRT) is a real-time path-planning algorithm [71]. RRT works by using a stochastic search over the body-centered frame of reference and expanding a tree through a random sampling of the configuration space [120]. RRT are suitable for handling dynamic problems with obstacles and differential constrains. Utilizing random samples from the search space, RRT

grows a tree that has its root at the starting location. The feasibility of connection with the tree is studied for each sample that is obtained. If the connection is feasible, the sample is added as a state to the tree. The length of the connection between the tree and the new sample is limited by a growth factor. This allows the tree to maintain an incremental growth.

1.3.3 Survey on Path-tracking Techniques for Mobile Robots

Path-tracking is the ability of a mobile robot to follow a given trajectory. Path-tracking is one of the important ability for a autonomous mobile robot. To be precise, path-tracking is the determination of the desired heading for a robot, utilizing the information such as path planned, the initial location of the robot and its heading angle in order to accurately track the desired path [125]. Methods to achieve path-tracking for autonomous mobile robots are generally grouped into two categories: geometric and control theoretic.

1.3.3.1 Geometric Methods for Path-tracking

Geometric methods for path-tracking generally use a virtual target point (VTP) to track the given paths. VTPs are look-ahead points on the desired path that assist the path-tracking algorithm. The path-tracking algorithm drives the robot towards VTP which eventually directs the robot onto the path. Following are some of the geometric approaches to path-tracking .

Carrot-Chasing Algorithm

Carrot-chasing algorithm or rabbit-chasing algorithm is one of the simplest path-tracking algorithm available [4]. The key concept of carrot-chasing algorithm is VTP. In carrot-chasing algorithm the VTP is called as carrot. It introduces VTP along the desired path and makes the robots to follow the VTP. The algorithm updates the heading direction of the robot to allow it to move towards the VTP. The robot will asymptotically approach and follow the desired path. Following is an overview of the steps involved:

1. Initialize VTP along the path to reach a waypoint.

2. Determine the distance between the current location of the robot and VTP. This distance is called the cross-track error.

3. Update the desired orientation and control input for the robot based on the cross-track error.

This process is continued once the robot reaches a waypoint. Usually a proportional controller is utilized to generate the control input. The selection of VTP plays an important role in achieving a stable path-tracking. If the distance between the VTP and the current position of the robot is very small, the robot tends to overshoot the VTP (especially when the robot is moving at higher velocities) leading to more cross-track error and resulting in an oscillatory performance in path-tracking.

Non-linear Guidance Law (NLGL)

The NLGL also utilizes VTP for path-tracking [102, 103]. NLGL will always produce a lateral acceleration that is appropriate to follow a circle of any radius. In order to decide on the control parameters to guide the robot towards the desired path, a circle is drawn with the robot at the centre of the circle. This circle should intersect the desired trajectory at two points. Based on the position of the robot with respect to the intersection points, one of them is utilized as VTP on the desired trajectory. Control parameters such as direction of acceleration is decided and lateral commands are generated using this VTP. The main advantage of NLGL is that it is suitable to be applied for any type of trajectory. NLGL utilizes an approximate PID control when the robot need to follow a straight path and anticipatory control when the desired trajectory is curved. The adaptive nature of NLGL ensures that the robot stability is not dependent on velocity, even in the presence of wind and other disturbances [103]. Usually the radius of the circle will be fixed after the first VTP is decided under the assumption that a circle with that radius will always intersect the desired trajectory. But this may not be true in all the scenarios. In conditions where the circle does not intersect the desired trajectory, there will be no VTP. On the other hand, if a large value is chosen as the radius of the circle the convergence time will be longer.

Pure Pursuit and Line of Sight-based Path Following (PLOS)

PLOS-based path-following is a combination of pure pursuit and Line of Sight (LOS) based navigation strategies [22, 69]. Pure pursuit approaches always work towards determining the curvature that will take a robot from its current position to a desired goal position. Waypoints along the desired trajectory or VTP qualify as desired goal positions. The goal positions are usually determined similar to the carrot-chasing algorithm. Pure pursuit algorithm defines a circle such that it passes through the goal point and the current position of the robot. Once the curvature required to take the robot from its current position to the next goal position is identified, LOS guidance laws are utilized to steer the robot towards its goal. The pure pursuit algorithm continually changes the target curvature, always pushing the goal point forward. The combination of these guidance laws will allows the robot to successfully follow the desired trajectory [125]. Though PLOS path-tracking method is stable it has certain limitations. The pure pursuit algorithm does not consider the dynamics of the robot while determining the curvature. This may result in scenarios where a sharp turn at high speed is to be done. In addition to this the robots tend to reach the desired paths at slow rates [22].

Vector-Field-based Path Following

In all the above mentioned methods, the path-tracking algorithms work to command the robot to a particular location on the desired trajectory. In vector-field-based path following, the fundamental objective is to be on the desired path rather than at a certain point at a particular time. Vector-field-based path following achieves path-tracking by constructing vector fields around the desired trajectory. Vector fields indicating the direction of flow from source to destination, for the robot to follow, are utilized in the Vector Field (VF) approach [91]. There are two modes of operation for this algorithm: one where the robot is away from the desired trajectory and the other where the robot is closer to the desired trajectory. The algorithm commands the robot to move towards the desired trajectory when the robot is far away. Once the robot enters a transition boundary (an user defined region within which the robot is considered near the

desired trajectory) the algorithm provides finer orientation control to accurately follow the desired path.

1.3.3.2 Control Theory-based Path-tracking Methods

Control theory-based path-tracking methods provide an alternative to the geometric path-tracking methods. Robustness to wind disturbances is one of the key advantages of using control theory-based path-tracking methods. Proportional Integral Derivative (PID) control based method is one of the commonly used control theory-based path-tracking method. A PID controller with feedforward controller is capable of outperforming a NLGL based controller [108]. Linear Quadratic Regulator (LQR), sliding mode control, model predictive control, backstepping control, gain scheduling theory, adaptive control and dynamic programming are some of the most common control theory-based path-tracking methods.

Linear Quadratic Regulator (LQR)

The LQR control-theory-based path-following algorithm utilizes optimal control theory to generate control effort for the robots. The final control effort generated minimizes robot's control effort by minimizing the cross track error and cross track error rate [137]. The control action for a LQR is given by

$$u = -\left(p\sqrt{|\ \frac{b}{b-p}\ |} + v_d\sqrt{2\sqrt{|\ \frac{b}{b-p}\ |} + q_{22}} \right) \tag{1.1}$$

where, p is the position of the robot, b is a boundary similar to the transition boundary in vector-field-based path following (explained in Section 4.1), v_d is the cross-track error velocity and q_{22} is the gain parameter of the Q matrix. Q matrix is defined as a second order matrix given by

$$Q = \begin{bmatrix} q_{11} & 0 \\ 0 & q_{22} \end{bmatrix} \tag{1.2}$$

where, $q_{11} =\mid b/(b-p) \mid$. Hence only b and q_{22} are the parameters to be tuned for LQR-based path-following algorithm. Manually tuning q_{22} parameter may be

troublesome at certain cases. For example a low q_{22} value will allow the robot to converge to the desired path quickly but with high control effort. On the other hand, a high q_{22} value will ensure that control effort is reduced while increasing the convergence time. Hence a value that achieves a trade-off between control effort and convergence time is to be decided.

Sliding Mode Control

Sliding mode control based path-following algorithm generally have fast response and good transient performance. The basic idea of a sliding mode controller is to force the robot to restrict its motion to a manifold called the sliding surface [45]. This is achieved by utilizing two different controls for both the sides of the robot, to direct the system trajectories towards the manifold. The main advantages of sliding mode control are their ability to decouple high dimensional problems into sub-tasks with lower dimensions and robustness with regard to parameter variations [45, 147]. The control law for the sliding mode controller is not continuous resulting in non-unique and non-existent solutions at certain time instances. This is avoided by utilizing a nonlinear control design.

1.3.4 Existing Vision-based Pedestrian Detection Techniques

Vision based human detection solutions are common and well researched upon. Human detection algorithms detect human body visible in a smaller portion of an image under different environmental conditions. Pedestrian detection is a difficult task as there could be wide range of possible pedestrian appearance due to changing articulated pose, clothing, lighting and background [31]. Pedestrian detection is made up of three parts: selection of Region of Interest (ROI), classification and tracking.

1.3.4.1 Region Of Interest Selection

ROI is usually decided based on more general low-level features or prior information about the scene and environment. A sliding window approach, where features are extracted from a rectangular portion of the image, is the simplest technique to identify ROI [23, 136]. Fast moving person detector utilizing Integral

Image and a classifier built using AdaBoost learning algorithm, a pioneer work in pedestrian detection, utilizes sliding window approach to identify ROI [136]. Histogram of Oriented Gradients (HOG) descriptors classified utilizing a linear SVM classifier is considered one of the best performing pedestrian detection algorithm utilizing sliding window approach [23].

The computational cost for sliding window approach is often too high leading to non-real-time image processing. There are many works available in the literature that deal with improving the processing time for sliding window-based approaches. Prior information about the target object class and known camera geometry is used for restricting the search space thereby reducing the computational cost. Prior information about the target object class includes application specific constraints such as assumption of flat-ground plane, ground plane based objects and common geometry of pedestrians (pedestrian height and aspect ratio) [43, 72, 88].

Identifying motion cues using object motion is another commonly used technique for ROI identification [90, 123, 153]. Background subtraction is utilized for static camera based pedestrian detection. Background subtraction generates a binary image containing information about the moving objects in the scene. As the name implies, background subtraction technique performs a subtraction between the current frame and the background model (containing static part of the scene) isolating information about the moving objects in the scene [90, 123, 153]. In situations where the camera is moving, deviation of the observed optical flow from the expected ego-motion flow field is computed to identify the ROI [32, 105].

1.3.4.2 Classification

Classification or the process of verification follows immediately after ROI identification in pedestrian detection. In the classification stage a sub-region of a given image is assigned to either pedestrian or non-pedestrian class, depending on appearance models utilizing various spatial and temporal cues. Appearance models for pedestrian verification are classified into generative and discriminative models depending on the corresponding class posterior probabilities. The major

difference between generative and discriminative models lies in the methods utilized for estimating the posterior probabilities for each of the classes.

Generative Models

Generative models utilize Bayesian approaches to infer the posterior probability for the pedestrian classes. In generative models the appearance of the pedestrian class are modelled in terms of their class conditional density function. Shape cues and shape cues combined with texture models are utilized for generating appearance models of the pedestrian classes.

1. **Shape models:**

   Shape models learn pedestrian shape models from shape contours in the images. Shape cues are useful in reducing variations in pedestrian appearance that arise due to changes in lighting or clothing. Shape models are discrete as well as continuous. Discrete approaches represent the shape using a set of exemplar shapes [31, 42, 43, 124, 131]. Modelling through exemplar shapes reduces false classifications but requires large amount of data to build a proper model. Continuous shape models learn class-conditional density through a set of training shapes from manual or automatic shape registration methods [8, 30, 46, 62, 88]. A prevalent continuous shape modelling approach is to use linear shape space representations that model class-conditional density as a single Gaussian [8]. A single linear shape space representation, in complex scenarios, lead to many intermediate physically implausible model instantiations. Non-linear shape space representation approaches are capable of handling complex scenarios but require a large number of training shapes as a result of the increased model complexity. Most common approaches break up nonlinear shape space into piecewise linear regions. Continuous generative models are capable of more complete shape representations through interpolation, compared to discrete shape models [31, 46, 88].

2. **Combined shape and texture models:**

   Combined shape and texture models combine separate statistical models for

shape and intensity variations into one compound parametric appearance model [20, 33]. Linear intensity models are built from shape-normalized examples. Iterative error minimization schemes are utilized in estimating shape and texture parameters for model fitting.

Discriminative Models

Discriminative models learn the parameters of the decision boundary between pedestrian and non-pedestrian classes from training samples and approximate the Bayesian maximum-a-posteriori decision. Discriminative classification techniques aim to determine an optimal decision boundary between pattern classes in a feature space [31]. Several classifiers such as multilayer neural networks, Support Vector Machines (SVM) and nonlinear SVM are utilized to construct classifiers that classify pedestrian and non-pedestrian classes. Feed-forward multilayer neural networks implement linear discriminant functions in the non-linearly mapped input pattern feature space [56]. Optimality of the decision boundary is assessed by minimizing the mean squared error (error criterion). SVM, a powerful tool to solve pattern recognition problems, maximize the margin of a linear decision boundary (hyperplane) to achieve maximum separation between classes [23, 60]. Non-linear SVM use polynomial or radial basis function kernels to map input samples into higher dimensional space. Non-linear SVM achieve improved performance with significant increase in computational cost and memory requirements [3, 101]. Machine learning meta-algorithms such as AdaBoost has also been utilized to construct strong classifiers as a weighted linear combinations of several weak classifiers [113]. This provides a whole cascade of increasingly complex pedestrian detectors with high processing speed.

Recently developed pedestrian detection techniques break down the complex appearance of the pedestrian class into manageable subparts [31, 35, 100]. In the first phase, a mixture-of-experts strategy identifies local pose-specific pedestrian clusters and a specialized expert classifier is trained for each subspace depending on the pose-specific pedestrian clusters [43, 143]. All the expert classifiers operate in parallel and the output of all the expert classifiers are combined utilizing

ensemble approaches such as majority voting, AdaBoost, trajectory-based data association and probabilistic shape-based weighing. In the second phase, a component-based approach is utilized to decompose pedestrian appearance into parts. The decomposed parts are either body parts such as head, torso and legs or codebook representations [117, 119]. In component-based approaches, the number of training samples required to cover a set of possible appearances is less compared to full-body detection approaches. Furthermore, component-based approaches are suitable for handling scenes with occlusions. Occlusion is a common problem that affects the detection performance of the pedestrian detectors. Component-based approaches handle occlusions if explicit interobject occlusion reasoning is incorporated into the model [76, 117, 143]. HOG–LBP detector, combining HOG and and Local Binary Pattern (LBP), is a human detection approach capable of handling partial occlusions [138]. A probabilistic pedestrian detection framework utilizing discriminative deep model for learning visibility relationship across part detections from multiple layers is able to handle occlusions more effectively [98].

1.3.4.3 Tracking Pedestrians

Tracking is the process of inferring trajectory-level information of a pedestrian detected. There are three major ways in which a detected pedestrian is tracked. Following are the three approaches:

1. A pedestrian detected is associated in each frame of the image based on geometry and dynamics of the pedestrian, without the help of pedestrian models.

2. Pedestrian appearance models are coupled with geometry and dynamics to track the pedestrian.

3. Pedestrian appearance models are combined with observation density, dynamics and probabilistic inference of the posterior state density to track pedestrians. Posterior state density is usually inferred by formulating it as a recursive filtering process. Particle filters are generally utilized for this

process as particle filters have the ability to closely approximate complex real-world multimodal posterior densities using sets of weighted random samples.

Tracking multiple pedestrians across images in real-time is a major problem in pedestrian tracking. A joint state-space involving a number of targets and their configurations inferred in parallel is the usual approach to track multiple pedestrians across images. The significant problem with this approach is the increase in dimensionality of the state space with increase in number of pedestrians. Grid-based likelihoods and sampling techniques such as Metropolis-Hastings sampling, partitioned sampling and annealed particle filters are some of the approaches to reduce the computational complexity. The second approach to track multiple pedestrians in real-time limit the number of objects to one per tracker. Multiple trackers are utilized to track multiple pedestrians in the image. Separate rules to separate neighborhood tracks and a method for initializing a track is to be defined for the second approach. Rules between multiple trackers are always formulated in terms of heuristics.

Real-time Pedestrian Detection

Most of the pedestrian detection algorithms utilize computationally expensive feature extractions or other computations that reduce their real-time performance. A real-time pedestrian detector [27] increased the speed of the state-of-the-art pedestrian detector by utilizing a sparsely sampled image pyramid to approximate features at intermediate scales. C^4 human detector utilizes CENTRIST features and human contour cues to achieve real-time performance without compromising accuracy [144]. Accelerated Feature synthesis a part-based pedestrian detection algorithm also achieves real-time running performance [73]. A pedestrian detection method that efficiently combines the flexibility of a part-based model with the fast execution time of a Random Forest classifier produces quasi real-time performance [87].

Deep Learning Models for Pedestrian Detection

In the recent years deep learning models for pedestrian detection have received

increased attention. A multi-stage contextual deep model simulating cascaded classifiers is successful in pedestrian detection [151]. A joint deep learning model for pedestrian detection is able to achieve a 9% reduction in average miss rate by jointly learning feature extraction, deformation handling, occlusion handling and classification components [99]. A Switchable Deep Network (SDN) automatically learns hierarchical features, salience maps and mixture representations of different body parts to isolate background clutter and reduce pose and viewpoint variations while detecting pedestrians [84]. Compared to deep learning models, feature based pedestrian detection techniques are computationally less expensive and easier for implementation on robots.

Biologically Inspired Methods for Pedestrian Detection

As visual object recognition is one of the fundamental cognitive task for human vision system, computer vision models that mimic human vision system have been actively researched upon. Computer vision models of human visual cortex have been applied for object recognition as well as pedestrian detection [52,118,152]. A mathematical model of human retina, with properties of spatio-temporal filtering, color-coding, non-linearity and the ability to sample, was developed to aid neuro-biologists in their research for artificial vision [9]. The retina model indicates the architectural as well as algorithmic principles and its advantages in developing efficient and fast for computer vision modules for low level image processing [11,48]. The current work in this thesis describes a retina-based biologically inspired model for pedestrian detection. The retina model is combined with existing state-of-the-art vision-based pedestrian detection algorithms and the detector with the best performance is utilized for human detection and avoidance system on an aerial and ground vehicle.

1.3.4.4 Non-vision-based Techniques for Pedestrian Detection

Pedestrian detection approaches based on laser range sensor and thermal cameras are also available in the literature. Laser-based pedestrian detection approaches to track a moving pedestrian from a static or a very slowly moving robot, detect humans by tracking the moving objects through a heuristic algorithm [77]. Two

lasers range sensors utilizing a combination of particle filters and probabilistic data association are also utilized for tracking multiple humans [115]. Human detection is also performed using algorithms that utilize fused data from a laser scanner and a video camera [10, 39, 75, 85]. A tilting laser scanner extracts body features from a human in front of the robot while the images from the camera are processed to detect the face of the human in front of the robot. Thermal camera is also utilized to detect humans [18, 53, 133]. Though these methods provide an alternative, vision-based pedestrian detection approaches are the widely used approaches for pedestrian detection.

1.4 Major Contributions

The contributions of this thesis are as follows:

- The main contribution of this thesis is the development of an interoperable multi-agent framework that allows easy usage of UAVs and UGVs for same missions. This is the first framework exploring the interoperability across UAVs and UGVs for same missions. Service Oriented Interoperable Framework for Robot Autonomy (SOIFRA) proposed and developed is a novel behaviour-based multi-agent framework . SOIFRA utilizes Belief Desire Intention (BDI) architecture for developing agents. SOIFRA is modular and made suitable for accommodating platform independent algorithms. Behaviours that are fundamental for autonomous operation of robots such as collision avoidance, real-time path-generation and path-following and pedestrian detection and avoidance are incorporated into SOIFRA.

- The development of a real-time path-generation and path-following algorithm (VDPGT) is another contribution. VDPGT is a geometry-based path-generation and path-following algorithm. VDPGT is platform independent and is useful for both UAVs and UGVs. The usability of VDPGT is demonstrated through realistic missions that require both the aerial and ground robots to collaborate and navigate in an unknown urban environ-

ment. VDPGT is incorporated into SOIFRA to provide path-generation and path-following behaviour.

- A human visual system-based pedestrian detection and tracking algorithm developed is another contribution. Mathematical models of Parvo and Magno channels of human retina are combined with existing state-of-the-art pedestrian detection techniques to achieve better pedestrian detection performances. The performance of the retina-based pedestrian detection algorithm is demonstrated through experiments with several pedestrian benchmark datasets. The developed retina model-based pedestrian detection algorithm has been incorporated into SOIFRA to provide pedestrian detection and avoidance ability. The modified framework is tested on both UAV and UGV on multiple scenarios.

1.5 Organization

This thesis focuses on developing a multi-agent framework that is interoperable across UAVs and UGVs. The overall thesis is organized as follows.

Chapter 2 provides an overview of SOIFRA, the interoperable multi-agent framework developed. The architecture of SOIFRA along with a detailed explanation of all the components of SOIFRA is provided. This chapter also provides an overview of the simulation and real-time environments utilized for the experiments. The configuration and specifications of the robots utilized for simulation and real-time experiments are listed.

Chapter 3 discusses about incorporation of collision avoidance behaviour into SOIFRA. A platform independent vision-based obstacle detection method developed for incorporation into SOIFRA is explained in detail. In addition to this, two methods for time-to-contact (TTC) estimation, an optical flow-based method and an expansion of object-based method are explained. The modifications required on SOIFRA to accommodate collision avoidance behaviour are explained in detail. Simulation and real-time experiments utilizing the modified framework are discussed to demonstrate the collision avoidance capability and modularity

of SOIFRA.

Chapter 4 deals with the incorporation of real-time path-generation and path-following behaviours into SOIFRA. A real-time path-generation and path-following algorithm (Vector Directed Path Generation and Tracking, VDPGT) developed, is presented and discussed in detail. The platform independent nature of VDPGT that is demonstrated through a series of simulation and real-time experiments on both aerial and ground vehicles is also discussed. A realistic mission that requires both aerial and ground vehicles to work cooperatively is discussed to demonstrate the usability of VDPGT. Path-generation and path-following agents that are developed and incorporated into SOIFRA are explained in detail. The modified framework with path-generation, path-following behaviours in addition to collision avoidance behaviour is utilized for a completing real-time and simulation missions.

Chapter 5 is about incorporation of pedestrian detection and pedestrian avoidance behaviour into SOIFRA. A human visual cortex model-based pedestrian detection and tracking algorithm developed is elaborated in detail. The performance of the retina model-based pedestrian detector developed is studied through several experiments on existing pedestrian benchmark datasets. The incorporation of pedestrian agent developed to perform pedestrian detection and avoidance is discussed in detail. Real-time experiments involving multiple scenarios, carried out to demonstrate the pedestrian detection and avoidance capability of SOIFRA, are also discussed.

Chapter 6 concludes this thesis. Possible future research directions are also discussed in the chapter.

Chapter 2

Interoperable Multi-agent Framework

"Miracles only occur for those with the determination to never stop trying"

– Emporio Ivankov

This chapter presents the conceptual and structural overview of Service Oriented Interoperable Framework for Robot Autonomy (SOIFRA) a multi-agent framework designed to be interoperable across UGV/UAV. SOIFRA is a behaviour-based framework utilizing BDI agents.

2.1 Background

In order to understand SOIFRA effectively, it is essential to understand some of the basic terminologies.

2.1.1 Autonomous Agents

An autonomous agent can be seen as a system that is capable of interacting independently and effectively with its environment in order to accomplish some pre-determined or self-generated tasks utilizing its own sensors and effectors [24]. Fig. 2.1 shows the basic architecture for an autonomous agent. The sensors receive input from the environment and transfer the data to the cognitive module. The

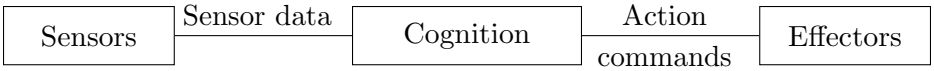


Figure 2.1: Basic architecture of an autonomous agent [24]

cognitive module decides the actions to perform based on the sensor information and commands the effectors accordingly.

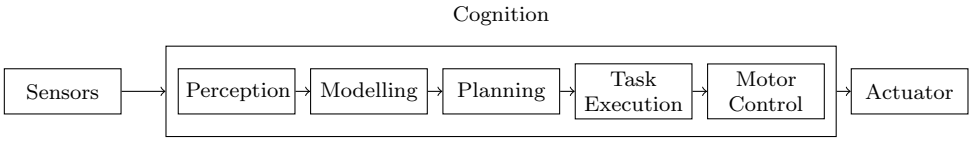
Autonomous agents are classified based on the the degree and manner of interaction with the real world. Situatedness and embodiment are used as features for classifying autonomous agents. Situated agents are agents that are situated in the world and they do not deal with abstract descriptions of the world. Embodied agents are the agents that experience the world directly and their actions have immediate feedback on the robots. Agents that have the least interaction with the world are pure computer simulations that are neither embodied nor situated. Agents that are situated but are non-embodied are classified as software agents. For example a monitoring-and-surveillance agent at NASA jet Propulsion Laboratory monitors inventory, planning and scheduling equipment ordering. Since the interaction of a monitoring-and-surveillance agent with the world is limited to sending and receiving messages, it is considered as non-embodied. On the other hand an industrial robot is an embodied and non-situated agent. Industrial robots are programmed to execute only a sequence of pre-programmed actions and they do not use the information about the current state of the environment to guide their behaviour [14, 24]. Autonomous mobile robots that perceive the environment and utilize the information obtained from environment to guide their behaviours are considered to be both situated and embodied.

Table 2.1: Types of autonomous agents

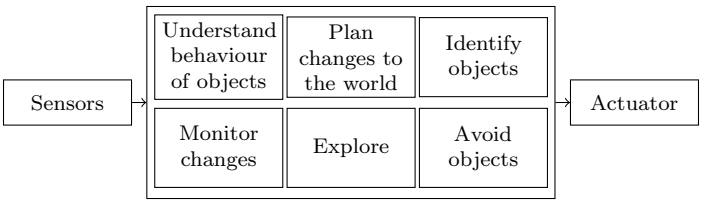
	Situated	Not situated
Embodied	Mobile robots	Industrial robots
Not embodied	Software agents	Computer simulations

2.1.2 Approaches for Designing Autonomous Agents

Agent architectures propose design methodologies for building autonomous agents. An architecture determines the ways to decompose complex problems into several sub-problems. To explain more clearly, an architecture determines the decomposition of an agent into several component modules and the interaction between the component modules [24]. The top-down approach and the bottom-up approaches are two of the major approaches for designing autonomous agents. In Fig. 2.2, sub-figs. 2.2a and 2.2b show the decomposition of a mobile robot control system using a top-down approach and bottom-up approach respectively. In the top-down approach an agent's cognitive capacity is modularized into several functional components. In the top-down approach the overall design of the architecture is finalized first and other components are added into the architecture later. On the other hand in a bottom-up approach simple behaviours, covering complete range from perception to action, are implemented first. Complex behaviours are added on top of single behaviours at a later stage.



(a) Top-down approach



(b) Bottom-up approach

Figure 2.2: Approaches for designing autonomous agents

2.1.2.1 Deliberative Approach

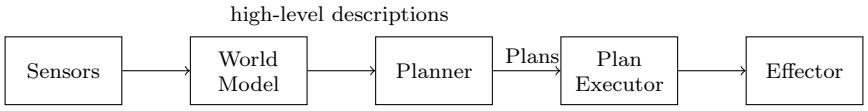


Figure 2.3: Architecture of a deliberative agent

The deliberative approach is one of the traditional approaches for designing agent architectures. Agents that are designed using deliberative approach are termed as deliberative agents. Fig. 2.3 shows the basic architecture of a deliberative agent. A deliberative agent contains an explicitly represented symbolic model of the world. The decisions about the choice of actions are determined through logical reasoning based on pattern matching and symbolic manipulation [142]. The world model is nothing but an internal description of the agent's external world as well as information about the capacity of the agent. World model is of two types [24]:

1. those that only describe the current state of the agent's environment and
2. those that include more general knowledge about other possible states and ways of achieving these states.

The planner utilizes the descriptions provided by the world model to generate plans for accomplishing agent's goals. Utilizing the information about the actions an agent can perform, the pre-conditions and effects of those actions on the environment, the initial and goal conditions, the planner searches through the operator sequences to identify a plan that will transform the initial state of the agent into the goal state. The world model and the planner together form the cognitive component of a deliberative agent architecture. The plans identified by the planner are passed on to the plan executor. The plan executor executes the actions detailed on the plans by utilizing several low-level routines of the effectors.

2.1.2.2 Reactive Approach

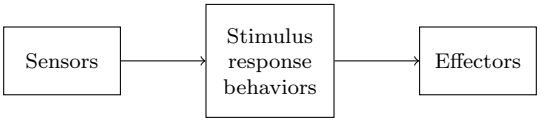


Figure 2.4: Architecture of a reactive agent

Reactive approach for designing agent architecture emerged based on the idea that most of our day-to-day activities consist of routine actions rather

than abstract reasoning. Reactive agents, instead of having a world model, have a collection of simple behavioural schemes that react to changes in the environment in a stimulus-response fashion. Fig. 2.4 shows the basic architecture of a reactive agent. In reactive approach behavioural decomposition receives more attention compared to the functional decomposition carried out by the deliberative approach. Usually in reactive approaches, the agents are constructed adopting a pure engineering approach. The simplest behaviours are added in first followed by the addition of more advanced behaviours. Reactive agents are mostly situated agents that act in direct contact with the real world without relying on the abstract descriptions included in the world model. In addition to this, a reactive agent should also be an embodied agent, so that the agent can react effectively.

2.1.2.3 Hybrid Approach

Hybrid approach for designing agent architectures utilize both high-level reasoning of deliberative approach and low-level reactive capabilities from reactive approach. Both deliberative and reactive approach to designing agents has disadvantages. While the deliberative agents are good at cognitive tasks such as planning and problem solving, they under perform in simpler tasks that require fast reaction with little to no deliberation. Reactive agents are good at solving simpler problems but lack versatility. Reactive agents under-perform in problems where knowledge about the world has to be obtained by reasoning or from memory rather than perception. As the weakness of reactive agents correspond closely with the strengths of a deliberative agent (and vice versa), hybrid approaches have less disadvantages. Hybrid approaches combine the reaction ability of reactive agents, necessary for routine tasks, with the power of deliberation, necessary for advanced and long term tasks. Fig. 2.5 shows a basic architecture of a hybrid agent.

2.1.3 Rational Agents

An autonomous agent is considered as a rational agent, if for each possible percept sequence, the agent selects an action that is expected to maximize its performance

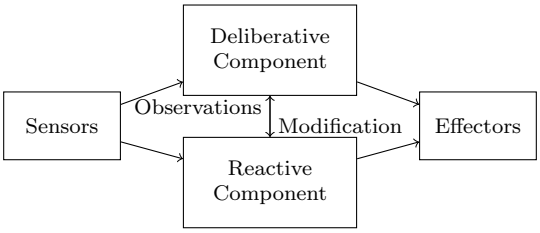


Figure 2.5: Architecture of a hybrid agent

measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has [112]. While performance measure is the criteria that determines how successful an agent is, percept sequence is the history of all the agent percepts upto that point in time. The rational behaviour of an agent is dependent on

- Performance measures that determine the degree of success,
- Percept sequences upto the current point of time,
- In-built knowledge about the environment and
- Actions that the agent can perform.

The problem a rationalized agent solves is characterized by Performance measure, Environment, Actuators and Sensors (PEAS). The action performed by a rational agent is considered as a right action, when that action causes the agent to be most successful for the given percept sequence.

Rational agents can be classified into four categories based on the different aspects of the environment in which the agents operate. Following are the major classifications of a rational agent

- Simple reflex agents
- Model-based reflex agents
- Goal-based reflex agents
- Utility-based reflex agents

2.1.3.1 Simple Reflex Agents

Simple reflex agents only act based on their current percept. There is no detailed knowledge base to associate and analyse every aspect of the sensor information received. Consider an autonomous vehicle following mission. Assuming only a vision sensor is utilized to maintain a distance to a vehicle in front, creating a complete and detailed look-up table for every pixel on the image frame is a highly complex task. Assuming 640 $\times$ 480 pixels of 8 bits of color and 8 bits of intensity information at the rate of 25 fps generates approximately 15 megabytes per second, for one hour, there needs to be $2^{60\times60\times15M}$ entries on the look-up table. On the other hand, if the information from the images are interpreted before decisions are made, there is no need for a complex look-up table. Simple condition-action rules (a rule that maps a state to an action) such as breaking when the vehicle in-front is breaking or accelerating when the vehicle in-front is accelerating will still complete the task. Fig. 2.6 shows the architecture of a basic reflex agent.

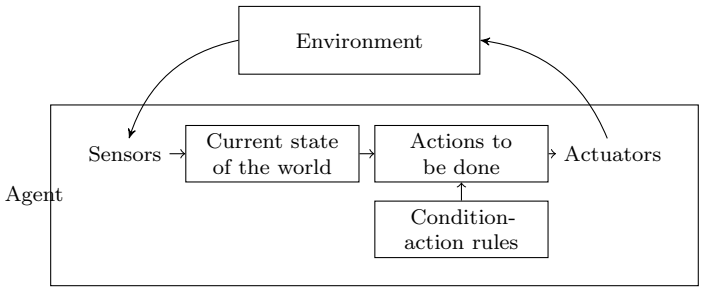


Figure 2.6: Architecture of a simple reflex agent

2.1.3.2 Model-based Reflex Agents

The actions of a simple reflex agent is correct only when correct decisions are made on the basis of the current percept. In most cases it is difficult to make correct decisions only based on the current percept. For the vehicle following problem, if the vehicle in front starts breaking, the vehicle in-front will appear closer in the image frame compared to the previous image frame. In addition to this if the rear light of the car in-front are not clearly visible then correct decisions

cannot be made. Hence it is necessary to maintain an internal state that gathers knowledge about the world in order to make correct decisions. The internal state should contain information about the changes in the world independent of the agent and information about the effects of agent's own actions on the world. Fig. 2.7 shows the architecture of a model-based reflex agent.

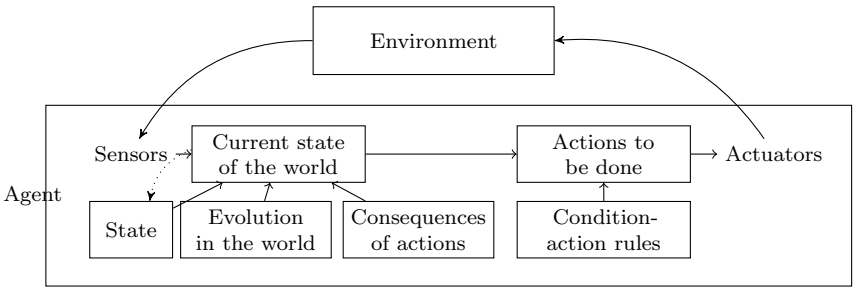


Figure 2.7: Architecture of a model-based reflex agent

2.1.3.3 Goal-based Reflex Agents

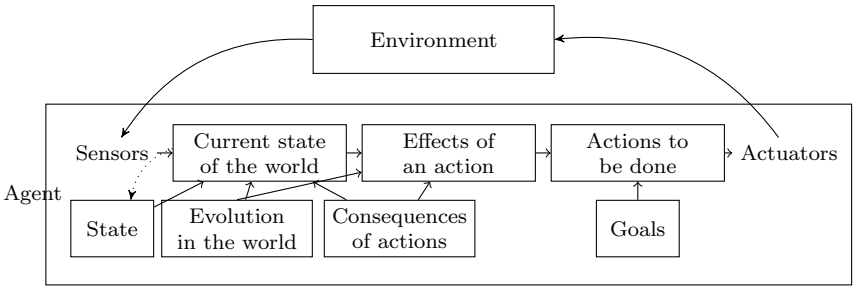


Figure 2.8: Architecture of a goal-based reflex agent

Knowledge about the current state of the environment alone is not sufficient for making correct decisions. In the vehicle following problem, if the current position of the vehicle in-front is lost, it will be difficult to locate the the vehicle in the subsequent image frames without sufficient information. If the information about the final objective or goal of the mission is known in prior, correct decisions can be made by combining these information with the knowledge from the internal state of the agent to generate correct actions that will satisfy the goal. These decisions are different from the condition-action rules as decisions are made considering the future. Fig. 2.8 shows the architecture of a goal-based reflex agent.

2.1.3.4 Utility-based Reflex Agents

There can be several possible actions that can be right for a given situation based on the information about the goal and knowledge of the internal states. In order to arrive at a proper decision in these type of situations, utility of the next achieved state comes into picture. Utility is considered as a function that maps a state onto a real number that describes the associated degree of effectiveness of an action. Utility provides a way in which the likelihood of success can be weighed up against the importance of the goals. The agent can use these real numbers to weigh the importance of achieving goals. Fig. 2.9 shows the architecture of a utility-based reflex agent.

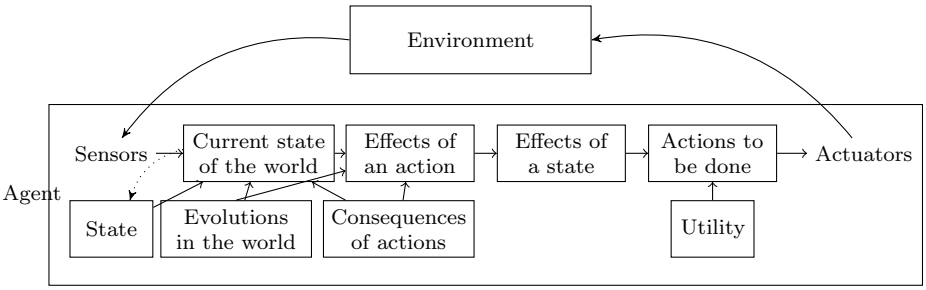


Figure 2.9: Architecture of a utility-based reflex agent

2.1.3.5 Belief Desire and Intention Agents

A BDI agent is a type of bounded rational agent. BDI design approach is an event-driven approach that provides both reactive and pro-active behaviour. It is generally accepted that BDI agents are at a level of abstraction closer to normal human experience. Mental attitudes such as beliefs, desires and intentions form the architectural components of a BDI system.

- Beliefs:

  Beliefs represent the informational state of an agent. It represents the beliefs about the world and beliefs about other agents and itself. Beliefs can be in the form of inference rules. Beliefs are different from knowledge base as belief represents only what an agent believes and it may not be true in all scenarios. Moreover belief of an agent may change based on future

events.

- Desires:

  Desires represent the motivational state of an agent. Objectives of an agent for a mission form the desires. There can be many desires for an agent. A goal is a desire that an agent is working towards at a given point of time in order to provide a restriction that at any point of time, the set of active desires must be consistent.

- Intentions:

  Intentions represent the deliberative state of an agent. Intentions are desires that an agent is working towards. An agent works towards achieving its desires by executing plans. A plan is a sequence of actions that an agent can perform to achieve one or more of its intentions.

BDI agent approach provides a mechanism for separating the activity of selecting a plan from the execution of currently active plans. This enables the BDI agents to balance the time spent on choosing the plan and doing the plan.

2.1.4 Summary

Autonomous agents interact independently and effectively with their environment to accomplish some pre-determined or self-generated tasks utilizing their sensors and effectors. Autonomous agents follow either deliberative, reactive or hybrid approaches for agent design. Deliberative approaches are effective in cognitive tasks that involve complex planning and problem solving while reactive approaches are effective in solving simple problems. Hybrid approaches combine the reaction ability of reactive approach and deliberative ability of deliberative approach for agent design.

A rational agent is a type of autonomous agent that always selects an action that is expected to maximize the performance based on the evidence provided by the percept sequence and the knowledge of the agent. Simple reflex agents, Model-based reflex agents, Goal-based reflex agents and Utility-based reflex agents are the four types of rational agents. Simple reflex agents act only based on their

current percept while model-based reflex agents act based on their knowledge about the current state of the environment and current percept. Goal-based reflex agents combine the information about the internal state of the agent and the information about the goal of their mission to generate actions. Utility-based reflex agents map a state onto a real number that describes the associated degree of effectiveness of an action to weigh the importance of achieving goals. BDI agents are bounded rational agents. BDI agents are at a level of abstraction closer to normal human experience. The BDI agent approach balances the time spent on choosing the plan and executing the plan, thereby improving the performance of agents.

2.2 Service Oriented Interoperable Framework for Robot Autonomy (SOIFRA)

SOIFRA framework developed is a behaviour-based interoperable framework for autonomous unmanned aerial and ground robots. SOIFRA is made up of deliberation, behaviour and execution layers. Goal-generator, planner-matcher and agents performing various services constitute the deliberation layer. Behaviour layer comprises of agent services, orchestration and choreography of services. Execution layer executes the actions carried out by agent services. Fig. 2.10 shows the architectural overview of SOIFRA.

2.2.1 Deliberative layer

The deliberative layer consists of goal-generator, planner-matcher and agents. The goal-generator block generates several sub-goals to accomplish a mission. Sub-goals are completed through pre-defined plans. For example, let us consider that the mission goal is to steer a robot towards a target in an unknown environment with obstacles. Goal-generator block generates two goals to achieve this mission; a goal to detect and avoid obstacles and a goal to reach the target. Each goal is further divided into sub-goals. For example, a goal to avoid collision is divided into obstacle detection and obstacle avoidance while the goal to reach a target

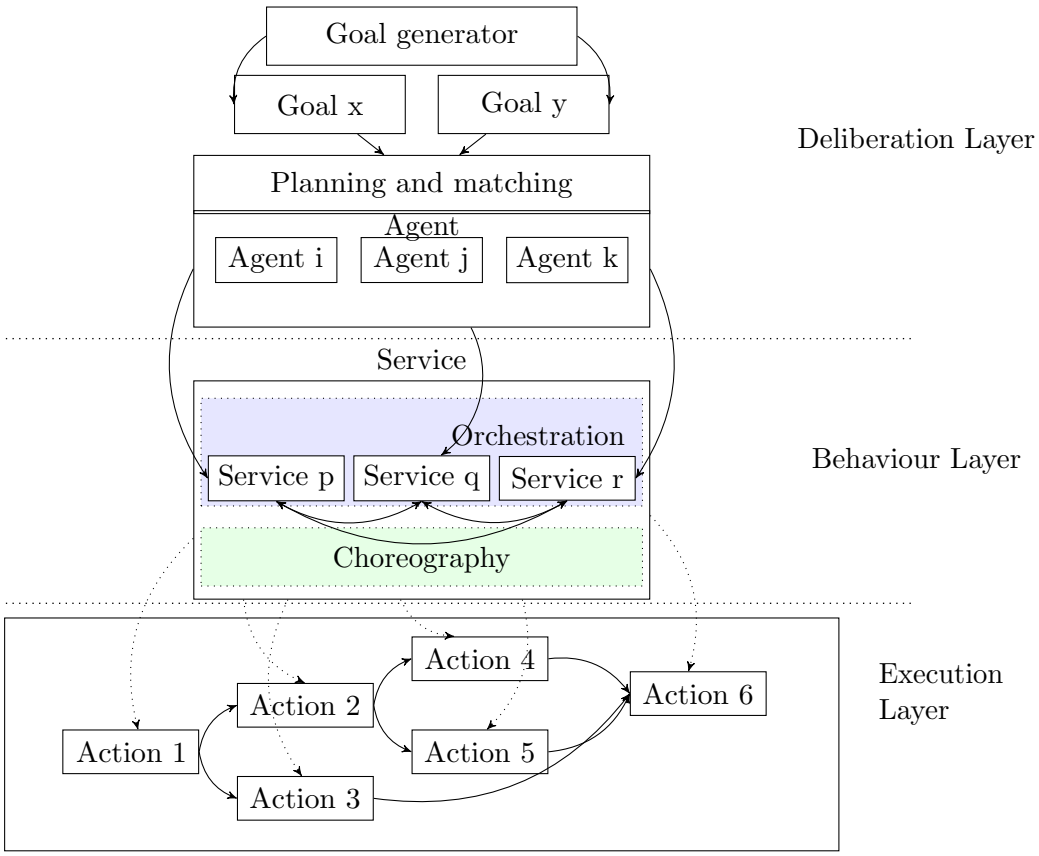


Figure 2.10: Architectural overview of the proposed framework (SOIFRA)

is divided into a navigation sub-goal. Sub-goals are then accomplished through the plans generated by planner-matcher module. Planner-matcher module helps to achieve a sub-goal by allocating agents to sub-goals based on the services offered by the agents. Agents, formed using a BDI approach can collaborate or act independently. Collaborative agents offer services to achieve one or more sub-goals, while non-collaborative agents offer services to achieve only one sub-goal. For accomplishing the mission, there needs to be a steering agent, providing services to achieve the sub-goals, obstacle avoidance and navigation and an Obstacle detection agent, providing obstacle detection service. Obstacle detection agent would operate as a non-collaborative agent that offer services to achieve one sub-goal, obstacle detection. Steering agent would need to operate as a collaborative agent as it supports both obstacle avoidance and navigation.

2.2.1.1 Planner-Matcher

The planner-matcher module in the deliberation layer (Fig. 2.10) allocates agents to sub-goals based on the plans generated. Built-in plans are used to generate plans based on the system ontology (Fig. 2.12). At any point, based on the prevailing states a competent pre-defined plan is retrieved from the ontological database and executed. Fig. 2.11 shows the process flow for agent goal allocation process performed by planner-matcher. Pre-defined plans are executed using a sequence of queries as shown below.

- Check if the robot has a predefined plan to complete a goal. If not, notify the planner that the system is not compatible. If it has a predefined plan, then proceed to the next query. The formal query statement associated is shown in Listing 2.1.

- Check if there are agents that are functionally capable of accomplishing a goal. If not, notify the planner, that the robot is functionally not competent. If it is functionally compatible proceed to the next query. The formal query statement associated is shown in Listing 2.2.

- Check if the services offered by agents are used by other sub-goals. If they are used by other sub-goals, check if the agents are collaborative. If not, intimate the planner that compatible agents are not available. If they are collaborative, request the agent identified for its services. The formal query statements associated is shown in Listing 2.3 and 2.4.

```
SELECT? Goal? Operation? Plan
WHERE {
        system:hasOperation(? Goal? Operation) ^
        system:hasPlan(? Operation)
}
```

Listing 2.1: Search query to retrieve a pre-defined plan from ontological database

```
SELECT? Agents? Functional_capacity? System_competence
```

```
WHERE {
        system:hasFunctionality(? Agent?
            ↪ Functional_capacity)
}
```

Listing 2.2: Search query to check the functional competence of the system

```
SELECT? Agents? Services_not_used
WHERE {
        system:hasFunctionality(? Agent?
            ↪ Functional_capacity)
}
```

Listing 2.3: Search query to check if the agent services are used other sub-goals

```
SELECT? Agents? Services_used
WHERE {
        system:hasFunctionality(? Agent?
            ↪ Collaborative_capacity)
}
```

Listing 2.4: Search query to check if an agent is a collaborative agent

2.2.1.2 Agent Structure

SOIFRA framework is composed of multiple distributed and independent agents for controlling the robot. In SOIFRA several agents are implemented to achieve a functionally modular agent framework. Autonomous operation of agents in a multi-agent framework requires knowledge about the environment. As world model in an autonomous agent is a representation of knowledge, the world model for agents in SOIFRA is made up of internal and external models. The internal model describes the self knowledge of an agent. Information about the internal operations and services of the agent form the internal model. Knowledge about surroundings and knowledge in social context represent the external model. Interactions among agents and effects of agent's services on the environment

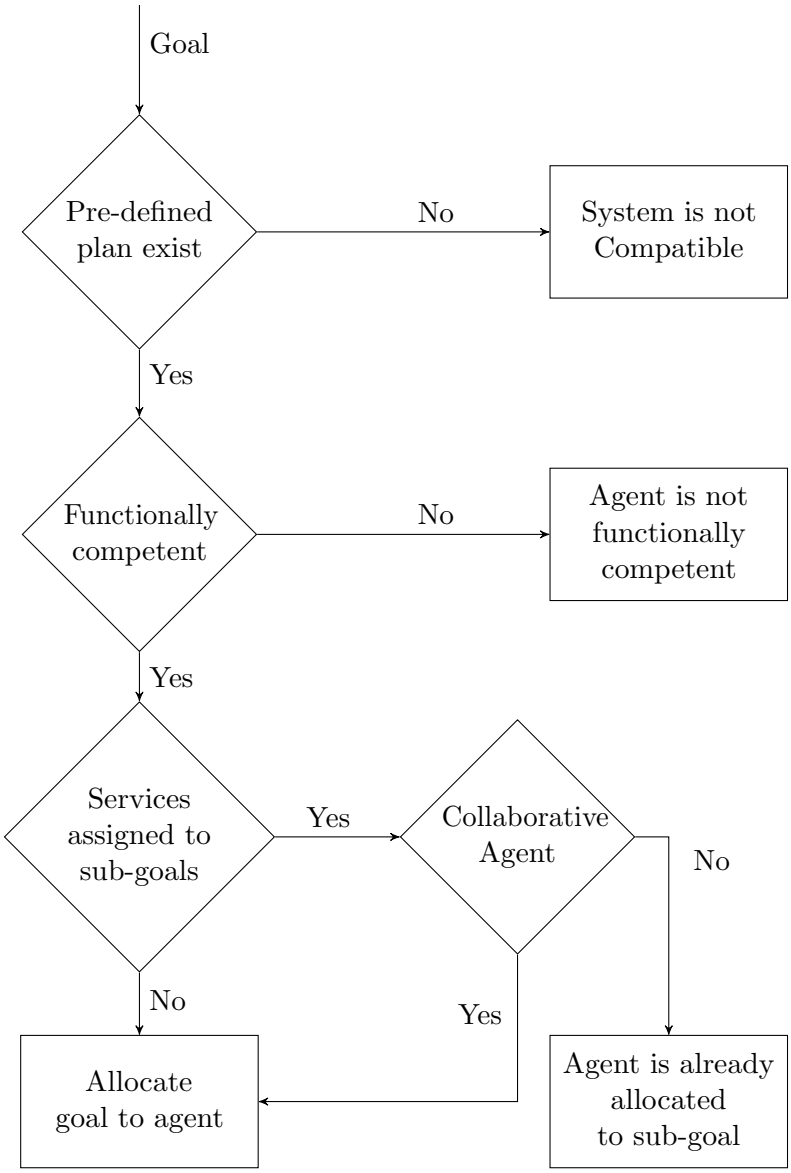


Figure 2.11: Process flow for agent goal allocation process performed by planner-matcher

form the external model. Information about events in the operating environment of an agent is obtained directly from sensors or agent's services. Internal and external world models form the belief of the agents in the Belief-Desire-Intention model (BDI) based framework developed. Desire of an agent is represented by the services and actions offered by the agent to carry-out the pre-defined plans allocated by the planner-matcher. The directions of the planner-matcher to achieve a sub-goal, represent agent's intentions.

Agents in the proposed framework are implemented in C++, utilizing Robot

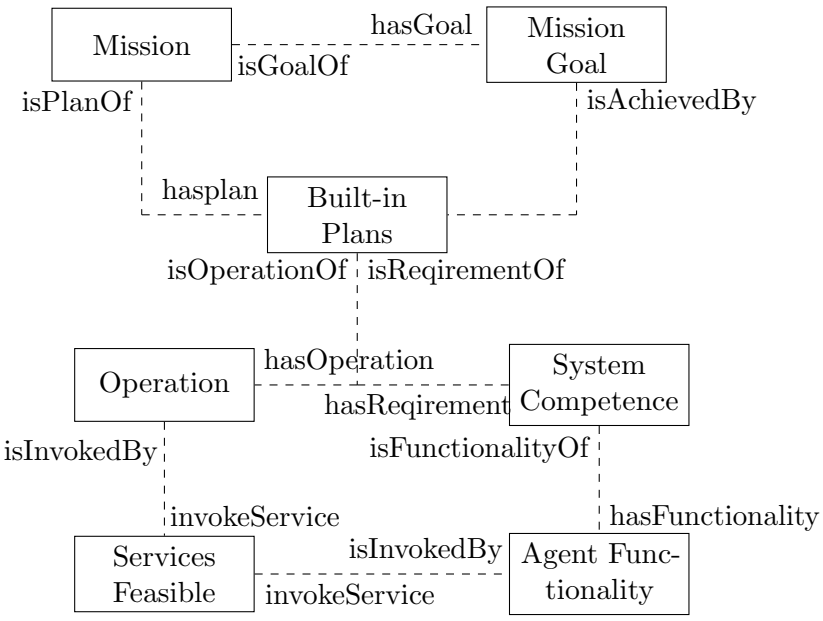


Figure 2.12: System ontology

Operating System (ROS) as the middle-ware. FIPA protocol is used for communication among agents. Agents register their services with a directory facilitator agent (DF agent), enabling the planner-matcher to allocate agents to sub-goal(s). This framework supports both collaborative and non-collaborative agents. Collaborative agents offer services that may be utilized to achieve single or multiple sub-goals. Non-collaborative agents achieve only one sub-goal. Sequences of operations by planner-matcher for allocating collaborative and non-collaborative agents to sub-goals are illustrated in Fig. 2.13. Collaboration among agents is achieved through a priority index. The Priority index of a collaborating agent is updated through the parameter server of ROS, utilizing the *dynamic_reconfigure* package. Deadlock condition occurs when agents sharing same resource prevent each other from accessing the resource while livelock condition occurs when agents that act in response to the actions of another agent wait for each other to complete. SOIFRA prevents these conditions through priority index. Using the same mission described above as example, obstacle avoidance and navigation agents are non-collaborative, achieving their respective sub-goals through collaboration with the steering agent. The steering agent assigns higher priority to the navigation agent when collision is not detected. Priority index of the steering

agent changes when a collision is detected and higher priority is assigned to the obstacle avoidance agent. Once the obstacle is avoided, priority index is updated, resulting in navigation agent attaining higher priority.

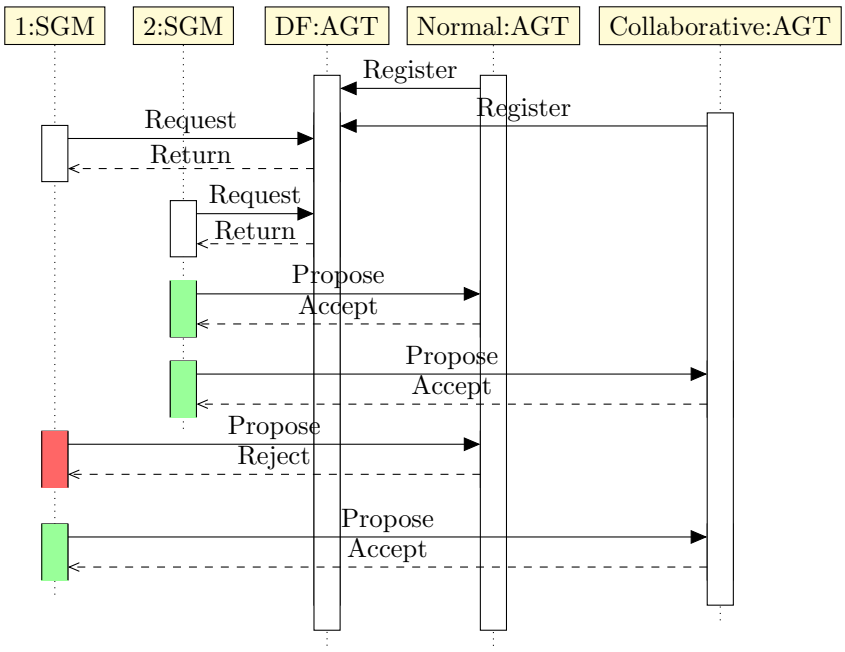


Figure 2.13: Illustration of goal allocation sequence. SGM and AGT refer to sub-goal planner-matcher and agent respectively. Agents (collaborative and non-collaborative) register with DF agent. Planner-matcher queries the DF agent to get a list of services offered by the agents. If the planner-matcher requires the service of an agent, it sends a request. The agent accepts or rejects it based on its availability and its collaborative nature.

2.2.2 Behaviour layer and Execution layer

Services performed by agents, and, orchestration and choreography of services form the behaviour layer. The execution layer is the lower most layer of SOIFRA framework. The execution layer comprises of actions performed by agents. Services depict the functional capacity of an agent. For example an obstacle avoidance agent will offer obstacle avoidance service, while a path-tracking agent will offer path-tracking service. Services are executed by combining various actions performed by an agent. Orchestration of services is the process where actions of an agent are combined into a service translating to the agent's behaviour. Orchestration of services plays an important role in achieving mission goal. Functional

decomposition of actions helps in improving agent modularity. For example the obstacle avoidance service can have multiple actions such as obstacle tracking and estimating time-to-contact (TTC) the obstacle. There is no restriction on the type of algorithm to be used for completing each of the action mentioned. For example, TTC an obstacle can be estimated through a variety of methods (optical flow-based method, expansion of an object-based method, etc). This algorithm to compute TTC can be replaced without affecting other actions of the obstacle detection agent or the structure of the framework. This allows to easily change different algorithms for same action. Each of these algorithms can be utilized for the same mission with less changes to the framework. Choreography of services represent communication among agents through messages. Actions are implemented as *rosnodes*, while choreography of services is implemented utilizing *rostopic* (named buses over which *rosnodes* exchanges messages). Some of the basic actions may be required by more than one agent. For example obstacle avoidance and navigation agents require the basic actions for controlling the velocity and orientation of the robot. This sharing of resources is achieved through collaboration among agents as explained in sub-section 2.2.1.2.

2.3 Simulation and Experimental Setup

ROS is used as a middle-ware in SOIFRA. ROS is a software framework for robot software development, providing operating-system like functionality on heterogeneous platforms. The architecture of ROS supports modularity that allows in developing platform independent algorithsm. Gazebo a multi-robot simulator is utilized for simulation experiments.

ROS consists of two parts, an operating system side and a package management side. The operating system part of ROS provides tools and libraries for obtaining, building, writing, and running code across multiple computers. ROS, initially developed by Stanford Artificial Intelligence Laboratory in support of the Stanford AI Robot STAIR project [106], is developed primarily at Willow Garage, a robotics research institute [107]. ROS provides standard operating system services such

as hardware abstraction, low-level device control, implementation of commonly used functionality, message passing between processes and package management. ROS is based on a graph architecture. Nodes in ROS act as processing center. Nodes may receive, post and multiplex sensor, control, state, planning, actuator and other messages. ROS is similar in some aspect to robot frameworks such as Player, YARP, Orcos, CARMEN, Orca, MOOS and Microsoft Robotics Studio as well as an Unix system. ROS also consists of a suite of user contributed packages (ros-pkg) that implement functionalities such as path-planning, Simultaneous Localization and Mapping (SLAM) and perception. ROS is not a real-time OS, though it is possible to integrate ROS in real-time with other software applications. ROS is released under the terms of BSD license, and is an open-source software. It is free for commercial and research use. The ros-pkg contributed packages are licensed under a variety of open source licenses.

2.3.1 Simulation Environment

Gazebo is a multi-robot simulator that offers ways to rapidly test algorithms, design robots for complex indoor and outdoor environments and perform regression testing using realistic scenarios. Gazebo is capable of simulating a population of robots, sensors and objects in a three dimensional world. Gazebo generates both realistic sensor feedback and physically plausible interactions between objects. It includes accurate simulation of rigid-body physics. By realistically simulating robots and environments, code designed to operate a physical robot can be executed on an artificial version. This helps to avoid common problems associated with hardware such as short battery life, hardware failures, and unexpected and dangerous behaviors. Dynamics simulation, Advanced 3D graphics, sensors, robot models, programmatic interfaces and TCP/IP communication are a few of the primary features offered by Gazebo. Over the years Gazebo has also been used for regression testing. The communication architecture is similar to ROS nodes. A simulated world usually publish body-pose updates and sensor generation while GUI consumes these messages to produce output. This mechanism allows for introspection of a running simulation and provides a convenient mechanism to

control aspects of Gazebo. Fig. 2.14 shows the architecture of Gazebo simulator.

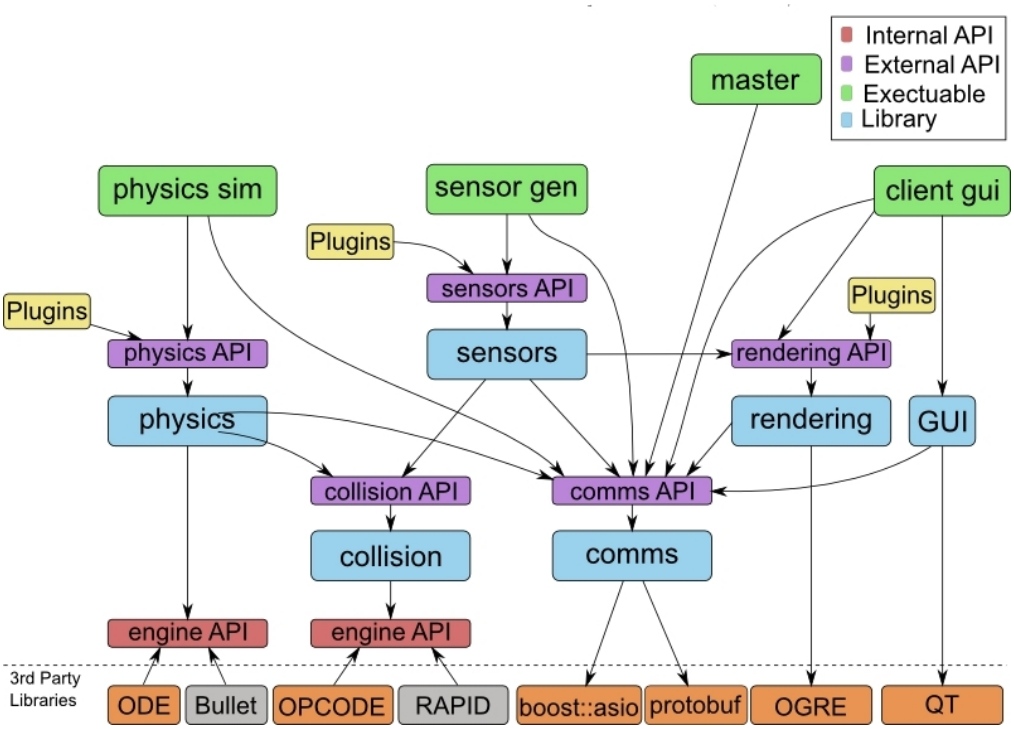


Figure 2.14: Architecture of Gazebo Simulator [51]

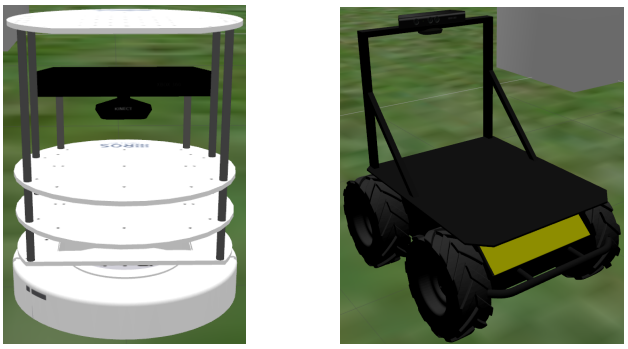
2.3.2 Robots used for Simulation Experiments

To prove the interoperability of SOIFRA, experiments are to be carried out utilizing different unmanned robots performing the same mission. Both unmanned aerial and ground vehicles are utilized for experiments. Turtlebot and Clearpath Husky are the unmanned ground robots used while Parrot AR Drone and Hector quadrotor are the aerial vehicles used. To achieve ROS integration with Gazebo *gazebo-ros-pkgs* package is utilized. *gazebo-ros-pkgs* provide wrappers and around the stand-alone Gazebo. This provides the necessary interfaces to simulate robots in Gazebo using ROS messages, services and *dynamic reconfigure*.

2.3.2.1 Ground Robots

Clearpath Husky and Turtlebot are the two ground robots used for simulation experiments. Both the robots are differential drive robots. They both utilize a Microsoft kinect for vision. Wheel encoders and Inertial Measurement Unit (IMU)

are both utilized to estimate the position of the robots in robot-frame. Fig. 2.15 shows the visualization of Clearpath Husky and Turtlebot utilized for simulation experiments. *husky-gazebo* packages are utilized for simulating Clearpath Husky in Gazebo environment while *turtlebot-gazebo* packages are utilized for simulating Turtlebot.

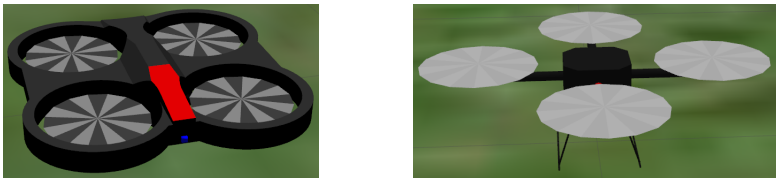


(a) Turtlebot (b) Clearpath Husky

Figure 2.15: Visualization of ground robots used for simulation experiments

2.3.2.2 Aerial Vehicles

Parrot AR Drone and Hector quadrotor are the two aerial vehicles utilized for simulation experiments. Hector quadrotor is simulated using *hector-quadrotor* package. *hector-quadrotor* contains packages that are related to modeling, control and simulation of quadrotor UAV systems. AR Drone is simulated using *tum-simulator*. *tum-simulator* mainly depends on *cvg-sim-gazebo, cvg-sim-gazebo-plugins* and *ardrone-autonomy*. *cvg-sim-gazebo* contains object models, sensor models, quadcopter models and flying environment information. *cvg-sim-gazebo-plugins* contain the gazebo plugins for the quadcopter model and plugins for sensors on the quadcopter such as IMU sensor, sonar sensor, GPS sensor, etc. *ardrone-autonomy* is the official ROS driver for Parrot AR Drone. Both the aerial vehicles utilize RGB camera plugin available in gazebo plugins for simulating camera.



(a) AR Drone (b) Hector Drone

Figure 2.16: Visualization of aerial robots used for simulation experiments

2.3.3 Robots used for Real-time Experiments

One aerial vehicle (Parrot AR Drone 2.0) and one ground robot (Turtlebot 2.0) is utilized for real-time experiments.

2.3.3.1 Turtlebot

Turtlebot is a low-cost differential drive robot based on ROS. Fig. 2.17 shows the image of Turtlebot 2.0. Turtlebot 2.0 consists of a Kobuki robot base that provides the base platform. The Kobuki base is then interfaced with a computer and is controlled using ROS packages. It also has Microsoft Kinect or ASUS Xtion PRO as vision sensor. Turtlebot is powed by a 4400 mAh Ni-Ion battery. Table. 2.2 provides the specifications for Turtlebot 2.0.

Figure 2.17: Turtlebot 2.0 used for real-time experiments

Table 2.2: Turtlebot 2.0 specifications

Size and weight	
External Dimensions	354 × 354 × 420 mm
Weight	6.3 kg
Wheels	76 mm
Ground Clearance	15 mm
Speed and Performance	
Max. Payload	5kg
Max. Speed	0.65m/s
Max. Rotational Speed	180^0/s
Sensors	
Vision Sensor	Microsoft Kinect
Encoders	25700 cps, 11.5 ticks/mm
Rate Gyro	110^0/s factory calibrated
Auxiliary Sensors	3×forward bump, 3×cliff, 2×wheel drop

2.3.3.2 AR Drone

AR Drone is a remote controlled quadcopter manufactured and sold by Parrot. The airframe of AR Drone is made up of nylon and carbon fiber parts. AR Drone has six degrees of freedom. A miniaturized inertial measurement unit is utilized to maintain roll, pitch and yaw stabilization. AR Drone's onboard computer is operated using Linux operating system. AR Drone can be connected to an external computer through a self-generated Wi-Fi hotspot that follows 802.11n standard. AR Drone is powered by a 11.1 volt lithium polymer battery that provides approximately 12 - 15 mins of flight time at a speed of 5 m/s. *ardrone-autonomy* is the official ROS driver for controlling an AR Drone. The ROS driver is based on the official AR-Drone SDK version 2.0.1. Fig. 2.18 shows the image of AR Drone 2.0 utilized for the real-time experiments and Table. 2.3 shows the specifications of AR Drone 2.0.

2.4 Summary

This chapter provides the conceptual and structural overview of the SOIFRA framework. In addition to presenting an overview of SOIFRA, the chapter also helps to understand some of the essential terminologies related to agents. The idea of autonomous agents and several approaches for designing autonomous

Figure 2.18: AR Drone 2.0 used for real-time experiments

Table 2.3: Specifications for AR Drone 2.0

Structure	
External Dimensions	57 × 57 cm
Weight	380 g (outdoor hull) / 420g (indoor hull)
Motors	
Motor	4× brushless, 14.5W, 28500rpm
Controller	4×MIPS AVS CPU
Camera	
Front Camera	1080p × 720p resolution, 30fps
Bottom Camera	640p × 480p resolution, 60fps
Rate Gyro	110^0/s factory calibrated
Electronics	
Processor	1 GHz 32-bit ARM Cortex A8
RAM	1 GB DDR2 at 200MHz
Sensors	3-axis gyroscope, accelerometer, magnetometer, Ultrasonic altitude sensor

agents such as deliberative, reactive and hybrid approaches are discussed. The concept of rational agents and various types of rational agents are outlined. Information about the simulation environment, robots utilized for simulation as well as real-time experiments are also discussed.

Chapter 3

Incorporating Collision Avoidance into SOIFRA

"Success is the sum of small efforts, repeated day in and day out"

– Robert Collier

SOIFRA is a behaviour-based multi-agent framework designed to be interoperable across unmanned aerial and ground vehicles and to accommodate generalized and platform independent algorithms. As collision avoidance is highly important in any autonomous unmanned robot mission, algorithms for obstacle detection and obstacle avoidance are incorporated into SOIFRA. This chapter elaborates on the collision avoidance functionality of SOIFRA while demonstrating the ability of SOIFRA to accommodate algorithms that are independent of platforms.

3.1 Collision Avoidance for SOIFRA

Collision avoidance is the ability of the robot to detect and avoid obstacles along its path. As SOIFRA provides a generalized framework that can support both aerial and ground vehicles, vision-based obstacle detection techniques are preferred because of their platform independent nature. Collision avoidance consists of two components obstacle detection and obstacle avoidance. Each component is elaborated separately.

3.1.1 Obstacle Detection in SOIFRA

Obstacle detection algorithm in SOIFRA utilizes Canny Contours, Hough Transform and Optical Flow algorithms. Once it is established that the robot is in collision course with the obstacle identified, the obstacle is tracked. The algorithms to detect obstacle(s) is explained in this section.

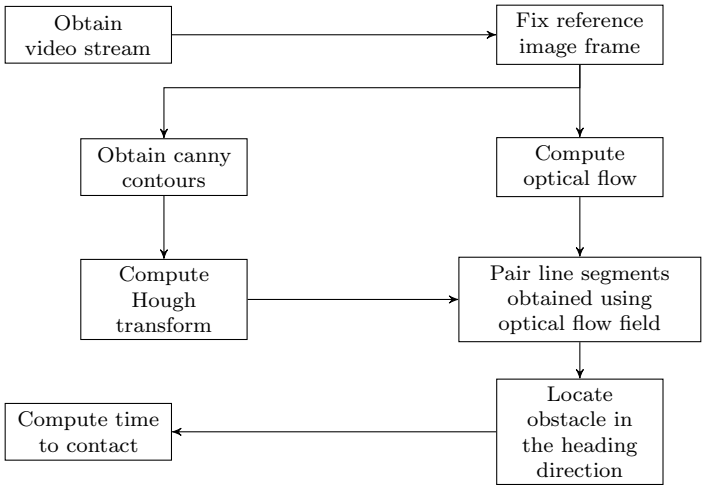
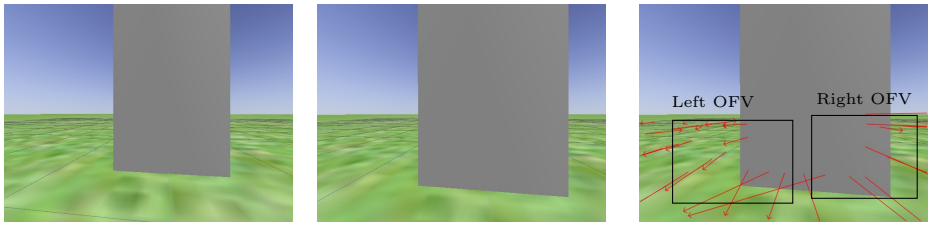


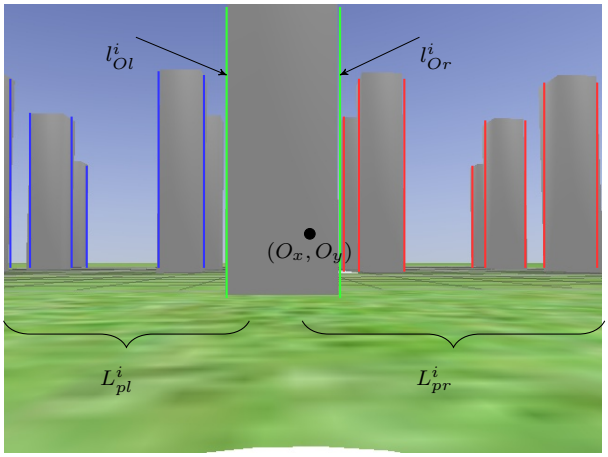
Figure 3.1: Process flow for obstacle detection

Obstacle detection is the first step in collision avoidance. The process flow for static-obstacle detection is shown in Fig. 3.1. The simulation and real-time environment, where the robots operate, are shown in Figs. 3.11 and 3.15 respectively. Canny contour, Hough transform and optical flow vectors are utilized to detect obstacles. Contours in the image are identified utilizing the Canny contour algorithm and line segments in the image are isolated using Hough transform. The obstacle detection algorithm (Algorithm 3.1) requires the optical centre of the camera (o_x, o_y) and image i at time t as its input. Let L^i be set of n line segments l^i_j, obtained through Hough transform for image i as shown in Fig. 3.2d (l^i_j is the j^{th} line segment obtained for image i). α^i_j is the angle associated with a line segment l^i_j and θ_t is the roll angle of the robot at time t of image i (θ_t, is zero for ground vehicles). Vertical edges of the objects in an image frame i are split into L^i_{pr} and L^i_{pl} based on its location with respect to the optical centre. If l^i_j is to the left of the optical centre it is grouped into L^i_{pl} and L^i_{pr} if it is to the right. l^i_{Ol} and l^i_{Or} are the line segments l^i_j of L^i_{pl} and L^i_{pr}, that

are the closest to the optical centre. OF^i_{Ol} and OF^i_{Or} are the optical flow vectors generated due to the motion of l^i_{Ol} and l^i_{Or} with respect to time. Lucas-Kanade Sparse optical flow algorithm is utilized for obtaining the optical flow vectors. Obstacle to be avoided is identified utilizing the concept that optical flow vectors from edges of an object do not intersect if the robot is in collision course with the obstacle (Fig. 3.3). If optical flow vectors in OF^i_{Ol} and OF^i_{Or} do not intersect, then l^i_{Ol} and l^i_{Or} are the identified as the edges of the obstacle to be avoided.



(a) Image i (b) Image $i+\tau$ (c) Flow of optical flow vectors



(d) Line segments l^i_j, obtained for an image i, are shown in red, blue and green colors. L^i_{pl} and L^i_{pr} include the line segments that are to the left (blue) and right (red) of the optical centre. l^i_{Ol} and l^i_{Ol} are the line segments (green) of L^i_{pl} and L^i_{pr}, that are the closest to the optical centre. Only line segments with $\alpha^i_j + \theta_t = 90^\circ$ are shown.

Figure 3.2: Fig.3.2c shows the optical flow vector generated from an obstacle as the robot moves towards the obstacle and left and right optical flow vectors generated from left side and right side of the obstacle. Fig.3.2a and 3.2b show the image frames utilized to generate the optical flow vectors shown in Fig.3.2c. Fig. 3.2d is an illustration showing the line segments l^j_i obtained for an image i.

Algorithm 3.1 Algorithm for Obstacle Detection

Input: Optical centre of the camera $O(O_x, O_y)$, image i at time t

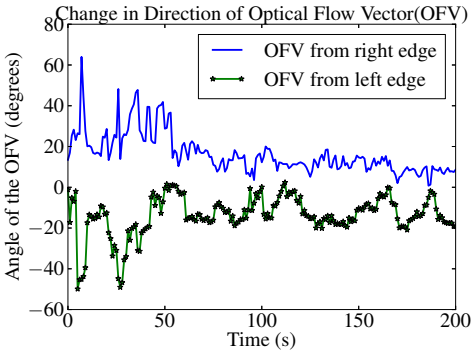
1: $L^i = \left\{l^i_j, \alpha^i_j\right\} = \{\{L_p\}, \{L_e\}\} \quad j = 1, \ldots n$, (Obtain all the line segments l^i_j and the angle associated with the line segments α^i_j in the image i using Hough transform)
2: **if** $\alpha^i_j + \theta_t = 90°$ **then**
3: **if** $l^i_{jx} < o_x$ **then**
4: $L^i_{pl} = \left\{l^i_j, \alpha^i_j\right\}$
5: **else**
6: $L^i_{pr} = \left\{l^i_j, \alpha^i_j\right\}$
7: **end if**
8: l^i_{Ol} is the l^i_j of L^i_{pl} closest to optical centre
9: l^i_{Or} is the l^i_j of L^i_{pr} closest to optical centre
10: OF^i_{Ol} and OF^i_{Or} are the optical vectors that are generated due to change in positions of l^i_{Ol} and l^i_{Or} with respect to time.
11: **if** Optical flow vectors in OF^i_{Ol} and OF^i_{Or} do not intersect **then**
12: Obstacle is detected between l^i_{Ol} and l^i_{Or}.
13: **else**
14: Obstacle is not detected.
15: **end if**
16: **else**
17: $L^i_e = \left\{l^i_j, \alpha^i_j\right\}$
18: **end if**

3.1.2 Obstacle Avoidance in SOIFRA

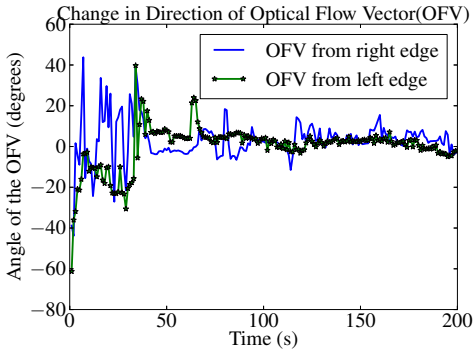
Obstacle avoidance is the second phase of collision avoidance. Once it is determined that the robot is in collision course with an obstacle detected, the distance to that obstacle from the robot is estimated continuously. Distance to the obstacle estimated based on optical flow and expansion of objects, are explained in this section.

3.1.2.1 Estimation of Time-to-contact

Time-to-contact (TTC), a quantitative measure, is useful for obstacle avoidance. The 3D information from optical flow fields of 2D images, extracted by time-to-contact, is utilized for obtaining distance to obstacle(s). In order to demonstrate that different algorithms can be utilized for same task without modifying the framework, two different time-to-contact algorithms are utilized. Time-to-contact is estimated utilizing optical flow and expansion of objects based methods. The error performances of both the methods for the operational environment under consideration are presented in Section 3.3.2. Once the time-to-contact is estimated,



(a) Change in direction of the optical flow vectors identified from the left and right edges of an obstacle in collision path of the robot



(b) Change in direction of the optical flow vectors identified from the left and right edges, when the robot is not in collision course with any obstacle.

Figure 3.3: Change in direction of optical flow vectors, associated with left edge (l^i_{Ol}) and right edge (l^i_{Or}) of the obstacle while a robot is moving forward. The optical flow vectors from the left and right edges do not intersect if the robot is in collision course with the obstacle.

the distance to the obstacle is obtained by,

$$Time\text{-}to\text{-}contact = \frac{Distance\ to\ the\ obstacle}{Velocity\ of\ the\ robot}. \tag{3.1}$$

3.1.2.2 Estimating Time-to-contact Utilizing Optical Flow (TTC-OF)

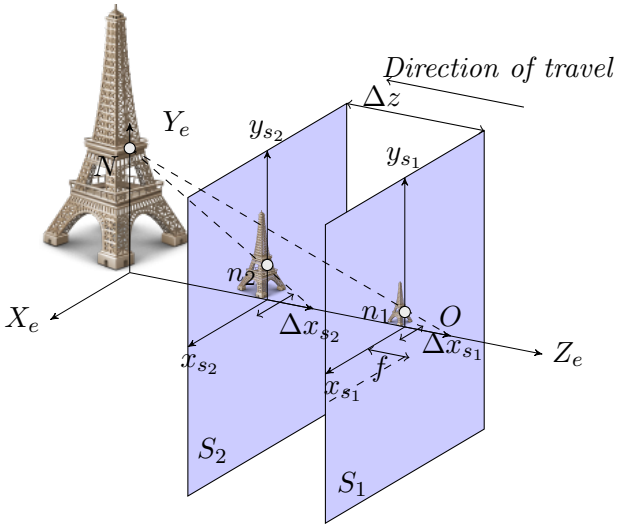


Figure 3.4: Projections of a point P onto image planes S_1 and S_2

Time-to-contact an obstacle is determined utilizing translational component of the optical flow. Time-to-contact estimation is independent of velocity of the

robot and distance to the surface of the obstacle [97]. Let f be the focal length of the camera on a robot, facing the direction of motion. For $N(X, Y, Z)$, a point on the obstacle, $n_1(x_{s_1}, y_{s_1}, z_{s_1})$ and $n_2(x_{s_2}, y_{s_2}, z_{s_2})$ are projections on image planes S_1 and S_2, at time instances t_1 and t_2 (Fig. 3.4). Robot undergoes translation along Z_e with a velocity $V = -\frac{dZ}{dt}$ over a distance $\Delta z = z_2 - z_1$, approaching the Focus of Expansion (FOE). From similar triangles,

$$\frac{y_{s_1}}{f} = \frac{Y}{Z} \Rightarrow y_{s_1} = f\frac{Y}{Z}. \tag{3.2}$$

Differentiating (3.2) with respect to time provides,

$$\dot{y_{s_1}} = f\left(\frac{\dot{Y}}{Z}\right) - fY\left(\frac{\dot{Z}}{Z^2}\right). \tag{3.3}$$

In (3.3) $\dot{Y} = 0$, as Y does not change with time. Substituting (3.2) in (3.3) and $\dot{Z} = -V$, we get

$$\dot{y_{s_1}} = -y_{s_1}\left(\frac{-V}{Z}\right) \Rightarrow \frac{y_{s_1}}{\dot{y_{s_1}}} = \frac{Z}{V} = TTC. \tag{3.4}$$

For a point N, on the obstacle, the distance from its projection, $p1$, on an image plane to the focus of expansion y_{s_1} and length of the optical flow vector $\dot{y_{s_1}}$, are required to estimate the time-to-contact (Equation 3.4). These calculated optical quantities are adequate in estimating the time-to-contact a point N on the obstacle.

FOE corresponds to the dynamic ambient optical array, which is a single point in space where all the optical flow vectors should emerge. Estimating FOE of an optical flow field is important in calculating the time-to-contact an obstacle (Equation 3.4). FOE is estimated by discrete, differential and least-squares-based methods, and performances of these methods are good only when the robot is in pure translation. Theoretically, FOE is the point of intersection of two optical flow vectors. In reality, noise and other errors arising from the steps in computing optical flow vectors affect FOE. In this work FOE is estimated using least squares solution (Equation 3.5 and Equation 3.6) of all the optical flow

vectors identified [130].

$$FOE = (A^T A)^{-1} A^T b, \tag{3.5}$$

$$A = \begin{bmatrix} a_{00} & a_{01} \\ a_{n0} & a_{n1} \end{bmatrix}, \qquad b = \begin{bmatrix} b_0 \\ b_n \end{bmatrix}, \tag{3.6}$$

where, for each pixel $n_i = (x, y)$ on the image, the associated optical flow vector $V = (u, v)$ gives $a_{i0} = v$, $a_{i1} = u$ and $b_i = xv - yu$.

3.1.2.3 Estimating Time-to-contact Utilizing Expansion of an Obstacle (TTC-EO)

Visual information obtained by monitoring the expansion of an object in visual field is utilized to obtain time-to-contact [15]. If expansion E is defined as the rate of growth of an object in the visual field of a robot, then time-to-contact is given by,

$$TTC \equiv \frac{1}{E}. \tag{3.7}$$

In Fig. 3.4, W is the width of the object, z_{s_2} is the distance between the lens (pinhole) and the object normal to the focal plane at time t_2 and, Δz is the distance traveled between t_1 and t_2. Δx_{s_1} and Δx_{s_2} are the width of the object projected onto image planes S_1 and S_2 respectively. Time-to-contact at t_1 is

$$TTC_1 = \frac{z_{s_2}}{\Delta z}, \Rightarrow z_{s_2} = TTC_1 * \Delta z. \tag{3.8}$$

From the pinhole camera model,

$$\Delta x_{s_1} = f\frac{W}{z_{s_2}}, \qquad \Delta x_{s_2} = f\frac{W}{z_{s_1}} = f\frac{W}{z_{s_2} - \Delta z}. \tag{3.9}$$

Expansion rate is given by,

$$E = \frac{\Delta x_{s_2} - \Delta x_{s_1}}{\Delta x_{s_1}} = \frac{\Delta x_{s_2}}{\Delta x_{s_1}} - 1. \tag{3.10}$$

Substitution of (3.8) and (3.9) in (3.10) results in

$$E = \frac{1}{TTC_1 - 1}. \tag{3.11}$$

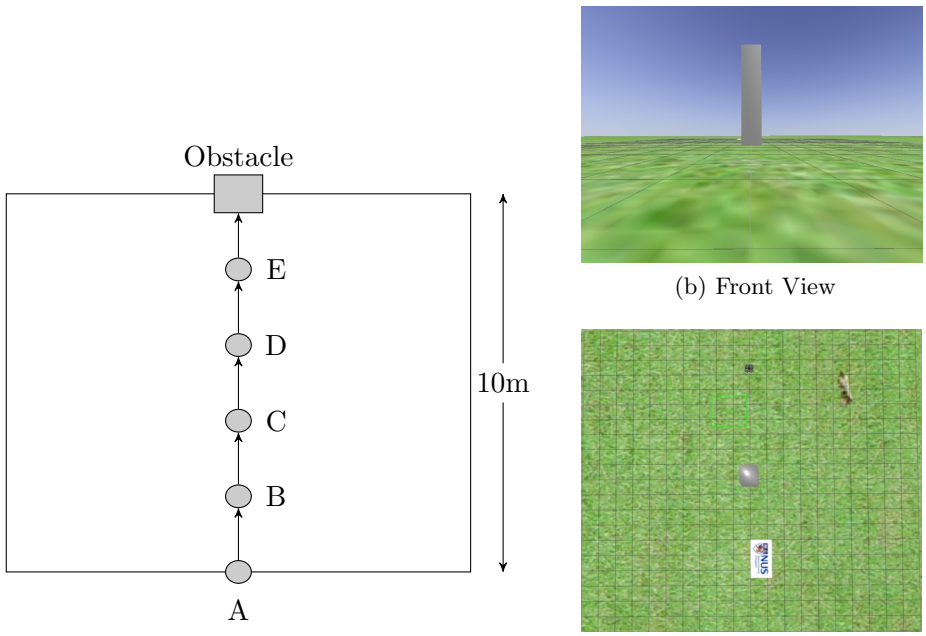
Re-arranging (3.11),

$$TTC_1 = (1 + \frac{1}{E})(t_2 - t_1), \ \textit{and}, \tag{3.12}$$

$$TTC_2 = \frac{1}{E}(t_2 - t_1), \tag{3.13}$$

where, TTC_2 is the time-to-contact at time t_2. Further explanation on implementations of TTC-OF and TTC-EO into SOIFRA are in Section 3.3.2. Experimental results and error performances of both the methods are shown in Fig. 3.16 and Table 3.5 respectively.

3.1.2.4 Estimating Operating Range for TTC-OF and TTC-EO



(a) Turtlebot

(b) Front View

(c) Top View

Figure 3.5: Layout of the simulation environment showing the starting positions of robot for different simulations. A, B, C, D and E are 2m from each other. Figs. 3.5b and 3.5c show the front and top view of the simulation environment respectively.

Theoretically, both TTC-OF and TTC-EO should be able to provide an

estimate of the distance to the obstacle, irrespective of the position of the robot. But both the methods operate only for a range beyond which the estimated distance will be error prone. In order to identify the safe operating range for TTC-OF as well as TTC-EO, experiments are conducted. A single aerial robot operating in a single obstacle environment is utilized for experiments. The aerial vehicle is made to move towards the obstacle with a constant velocity, starting from different locations in the environment. In Fig.3.5, Fig. 3.5b and Fig. 3.5c shows the top and front views of the operating environment. Fig. 3.5a shows layout of positions where the aerial vehicle starts for different experiments. Fig. 3.6 shows the error in distance to the obstacle computed and the change in length of the optical flow vector for each experiment. In order to get a clear picture of the error in distance to the obstacle estimated, mean error error metric is utilized. Mean error in distance to the obstacle is computed using,

$$Mean\ error = \frac{\sum_{t=i}^{t=k}(d_{true} - \hat{d})}{No.\ of\ elements\ in\ the\ interval} \tag{3.14}$$

where, d_{true} is the true distance to the obstacle and $\hat{d}$ is the distance to the obstacle estimated using time-to-contact computed and i and k denote the starting and ending time of the interval between which the mean error is computed. Time interval between i and k is approximately 5s. Table. 3.1 tabulates the mean error in distance to the obstacle estimated utilizing TTC-OF.

It can be seen clearly from Fig. 3.6 that time-to-contact computed will have a lot of fluctuations during the initial period and it settles down after some point in time. If the plots in Fig. 3.6 are analysed, it can be found that the time taken for this stabilization decreases as the distance to the obstacle decreases. If the corresponding length of the optical flow vector plots are analysed, it can be found that the approximate length of the optical flow vector around the stabilization time for time-to-contact is 150px. Table 3.2 shows the time taken for the optical flow vector to reach 150px when starting from the locations shown in Fig. 3.5a.

It can be seen clearly from table 3.2 that the stabilization time for time-to-contact decreases as the distance to the obstacle decreases. The reason for this

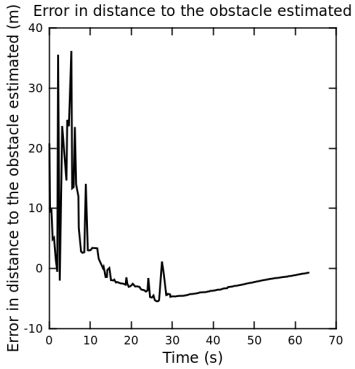
Table 3.1: Mean error of the distance to the obstacle estimated utilizing TTC-OF. A, B, D and E represent the starting locations of robots (shown in Fig.3.5a)

Time(s)	Mean Error(m)			
	A(10m)	B(8m)	D(4m)	E(2m)
0-5	14.77505	22.87588	15.80010	6.23388
6-10	9.83226	14.50319	4.26219	-0.71799
11-15	0.62443	13.17127	0.48042	–
16-20	-2.43871	-0.73166	-0.69162	–
21-25	-3.75778	-4.17175	–	–
26-30	-3.89701	-3.52019	–	–
31-35	-4.31287	-2.79364	–	–
36-40	-3.74661	-1.82093	–	–
41-45	-3.03261	-1.17550	–	–
46-50	-2.29101	-0.82194	–	–
51-55	-1.61453	–	–	–
56-60	-1.03233	–	–	–
61-65	-0.69896	–	–	–

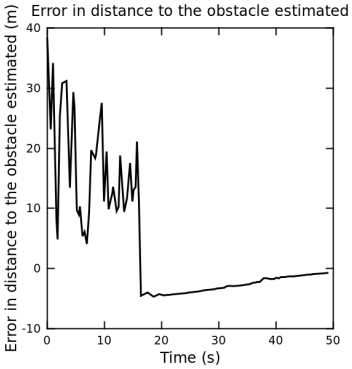
Table 3.2: Approximate time taken for the length of the optical flow vector to reach 150px

Starting distance from the obstacle (m)	Time (s)
10	30
8	20
4	10
2	2

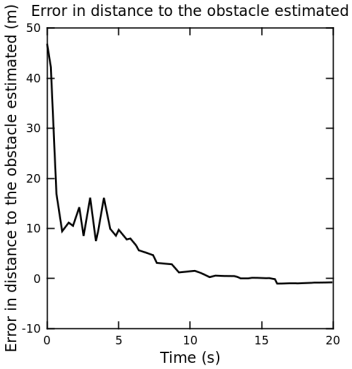
can be understood by observing Eqn. 3.4 closely. From Eqn. 3.4 it can be found that the length of the optical flow vector is in the denominator and when the length of the optical flow vector is small, time-to-contact increases. Though the aerial vehicle is moving with a constant velocity throughout, the time taken for the length of the optical flow vector to reach 150px varies. When the obstacle is far away from the aerial vehicle, the relative motion of the obstacle in the image plane is too small. Similarly when the aerial vehicle is close to the obstacle the relative motion of the obstacle in the image plane is large compared to the previous case. This is the main reason for the difference in stabilization time for the time-to-contact estimation. Based on these results it can be safely established that the time-to-contact estimated using TTC-OF will be more accurate if the robot is operated in the region where the true distance to the obstacle is less than 5m.



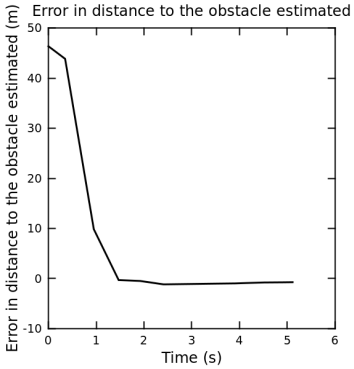
(a) Error in distance computed when robot starts from A



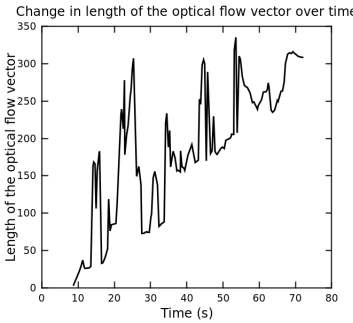
(b) Error in distance computed when robot starts from B



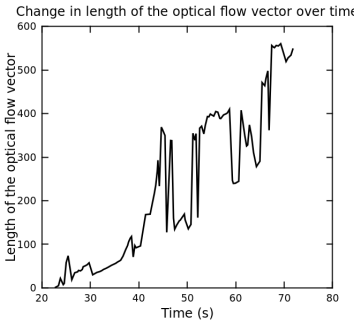
(c) Error in distance computed when robot starts from D



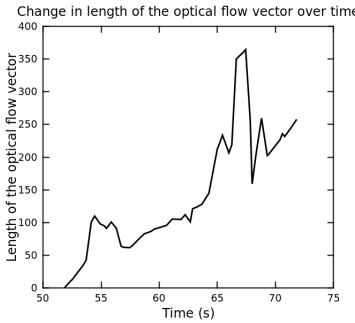
(d) Error in distance computed when robot starts from E



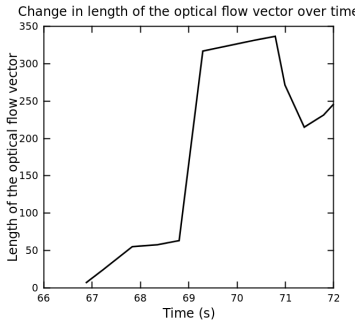
(e) Change in length of the optical flow vector. Robot starts from A



(f) Change in length of the optical flow vector. Robot starts from B

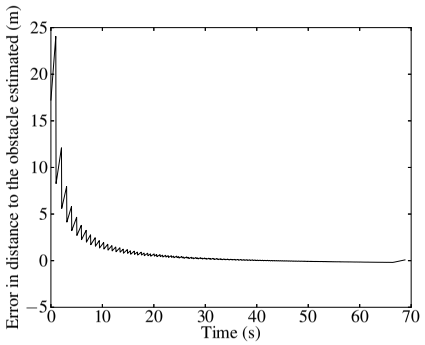


(g) Change in length of the optical flow vector. Robot starts from D

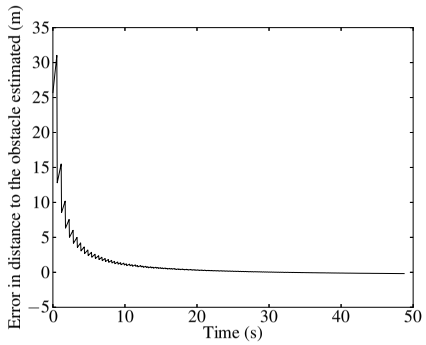


(h) Change in length of the optical flow vector. Robot starts from E

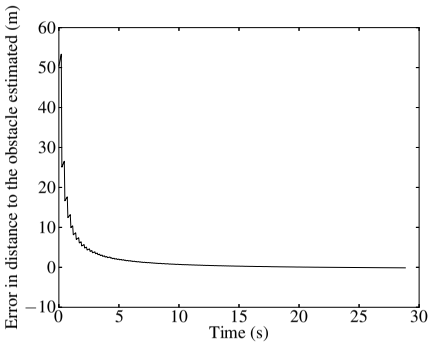
Figure 3.6: The plots show the change in length of optical flow vector and error in distance to the obstacle estimated utilizing TTC-OF for various positions (A, B, D and E) shown in Fig. 3.5a. The length of the optical flow vectors is measured in pixels



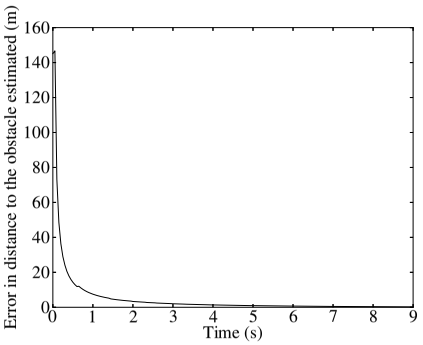
(a) Error in distance computed when robot starts from B



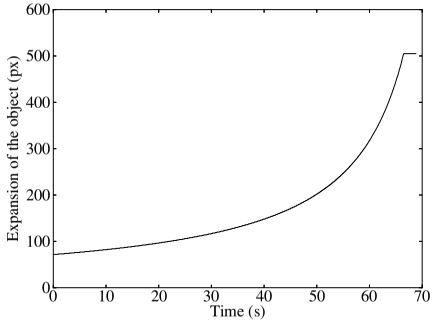
(b) Error in distance computed when robot starts from C



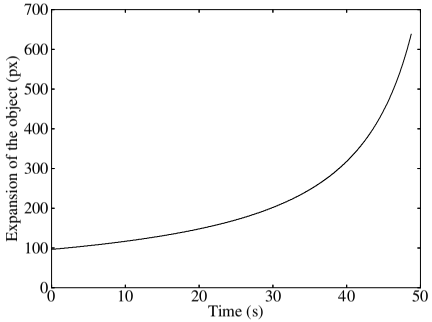
(c) Error in distance computed when robot starts from D



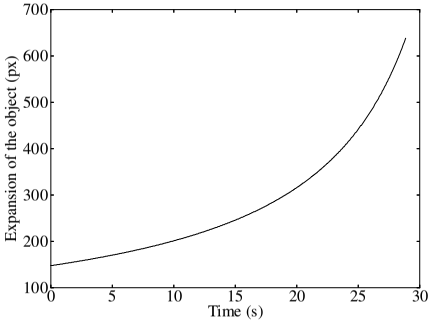
(d) Error in distance computed when robot starts from E



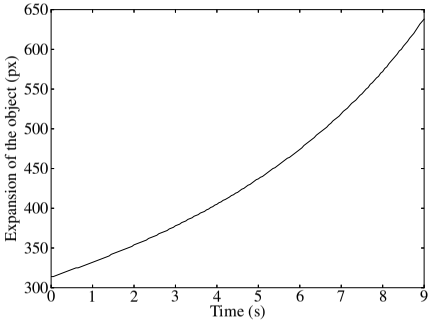
(e) Change in length of the optical flow vector. Robot starts from B



(f) Change in length of the optical flow vector. Robot starts from C



(g) Change in length of the optical flow vector. Robot starts from D



(h) Change in length of the optical flow vector. Robot starts from E

Figure 3.7: The plots show the change in length of optical flow vector and error in distance to the obstacle estimated utilizing TTC-EO for various positions (B, C, D and E) shown in Fig. 3.5a. The length of the optical flow vectors is measured in pixels

Table 3.3: Mean error of the distance to the obstacle estimated utilizing TTC-EO. B, C, D and E represent the starting locations of robots (shown in Fig.3.5a)

Time(s)	Mean Error(m)			
	B(8m)	C(6m)	D(4m)	E(2m)
00.0 - 5.0	9.365	9.005	8.836	8.836
05.0 - 10.0	2.368	1.753	1.204	0.501
10.0 - 15.1	1.243	0.825	0.472	–
15.1 - 20.1	0.739	0.439	0.168	–
20.1 - 25.1	0.463	0.223	-0.000	–
25.1 - 30.2	0.286	0.086	-0.096	–
30.2 - 35.2	0.160	-0.009	–	–
35.2 - 40.2	0.071	-0.078	–	–
40.2 - 45.3	0.001	-0.132	–	–
45.3 - 50.3	-0.053	-0.168	–	–
50.3 - 55.4	-0.098	–	–	–
55.4 - 60.4	-0.135	–	–	–
60.4 - 65.5	-0.166	–	–	–
65.5 - 68.8	-0.088	–	–	–

Table 3.4: Approximate time taken for the error to reduce to 0.5m using TTC-EO

Starting distance from the obstacle (m)	Time (s)
8	22.7
6	16.75
4	11.85
2	6.8

In Fig. 3.7, sub-figs. 3.7a, 3.7b, 3.7c and 3.7d show the error in distance to the obstacle estimated using TTC-EO, when the robot starts from B, C, D and E (shown in Fig.3.5a) respectively. From these plots it is apparent that the error in distance to the obstacle estimated is large initially and reduces to near zero after some time. This time taken for the error to reduce has a direct relation with the distance of the obstacle from the robot. If the vehicle starts farther from the obstacle the time taken for the error to reduce to 0.5m is large and the time reduces as the robot moves closer to the obstacle, as shown in Table. 3.4. This is because, the change expansion of the object when the robot is far away from the obstacle is minimal compared to the change in expansion of the object when the robot is near the obstacle. When the robot is near the obstacle, even smaller translation towards the obstacle produces larger value of expansion, while when the robot is further away from the obstacle, the robot has to move significantly

to generate the similar expansions. Based on the results obtained (Table. 3.3, Table. 3.4 and Fig. 3.7) it can be verified that the safe operating region for the robot while using TTC-EO would be less than 6m similar to TTC-OF.

3.2 SOIFRA with Collision Avoidance Behaviour

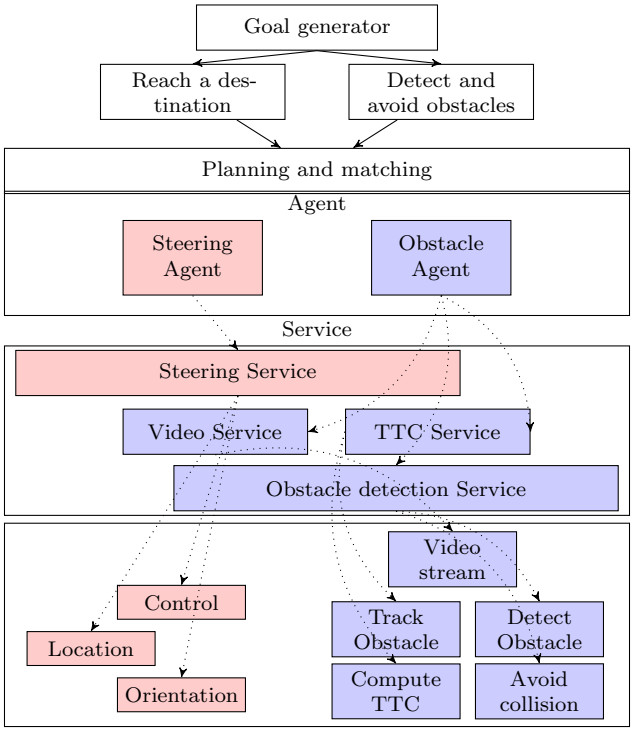


Figure 3.8: Architectural overview of SOIFRA with collision avoidance.

Architecture of SOIFRA, a behaviour-based interoperable framework developed for autonomous unmanned aerial and ground vehicles is explained in Section 2.2 of Chapter 2. This section explains the incorporation of collision avoidance behaviour into SOIFRA. SOIFRA consists of goal-generator, planner-matcher and agents in the deliberation layer, agent services in the behaviour layer and the actions carried out by agent services on the execution layer. Though collision avoidance is one of the important functionality for an autonomous robot, collision avoidance can only be a sub-goal. Collision avoidance cannot be the only goal for a robot as collision avoidance comes into picture only when the robot is moving (under the assumption that only static obstacles are present in the environment). Hence collision avoidance should coexist as a sub-goal with

another sub-goal. In order to understand the incorporation of collision avoidance into SOIFRA effectively, a mission where the robot has to move towards a target region while avoiding static obstacles is considered. Fig. 3.8 shows the architectural overview of SOIFRA for the mission specified.

The overall goal for the mission is split into two sub-goals: a sub-goal to detect and avoid obstacles and another sub-goal that ensures that the robot reaches its target destination. Two agents are developed to address these sub-goals. An agent that navigates the robot and an agent that performs obstacle detection and avoidance. Let Str:AGT represent the steering agent that controls the navigation of the robot and Obs:AGT represent the obstacle agent that is responsible for obstacle detection and avoidance. Str:AGT provides steering service (str:SRV) while Obs:AGT provides video service (video:SRV), TTC service (TTC:SRV) and obstacle detection service (det:SRV). Str:AGT, through str:SRV, carries out actions that drive the robot (ctr:ACT), and measure the position (pos:ACT) and orientation (ori:ACT) of the robot. Obs:AGT executes the action to obtain the video stream (video:ACT) through video:SRV; executes the actions to detect (det:ACT) and track obstacle(s) (track:ACT) in collision coarse with the robot through det:SRV. Obs:AGT also executes the actions to compute TTC (TTC:ACT) and collision avoidance (avd:ACT) through TTC:SRV.

Let PS:ROS and topic:ROS denote the parameter server and *rostopic* respectively. The obstacle agent, upon initiation, starts the actions for video service and obstacle detection service and publishes the control velocity for the robot. The steering agent subscribes to the velocity commands from the obstacle agent. When an obstacle is detected, the obstacle agent initiates the time-to-contact service and updates the control velocity for the robot. If the time-to-contact service estimates that the distance to the obstacle is less than the critical distance λ, (the minimum distance to the obstacle within which action must be taken to avoid an obstacle) the obstacle agent updates the ROS parameter server. As a result, the priority index of the steering agent is updated resulting in initiating obstacle avoidance process. If the distance to the obstacle is more than the critical distance, the robot continues to follow its previous path. The critical

distance, determined based on the size and linear velocity of the robot, is fixed at 2.5m for the current mission. Once the obstacle detected is avoided, the steering agent informs the obstacle agent that the obstacle is avoided and the steering agent allows the robot to continue to follow a straight path. This feedback allows the obstacle agent to restart the det:SRV. Fig. 3.9 shows these sequences of operations for obstacle detection and avoidance.

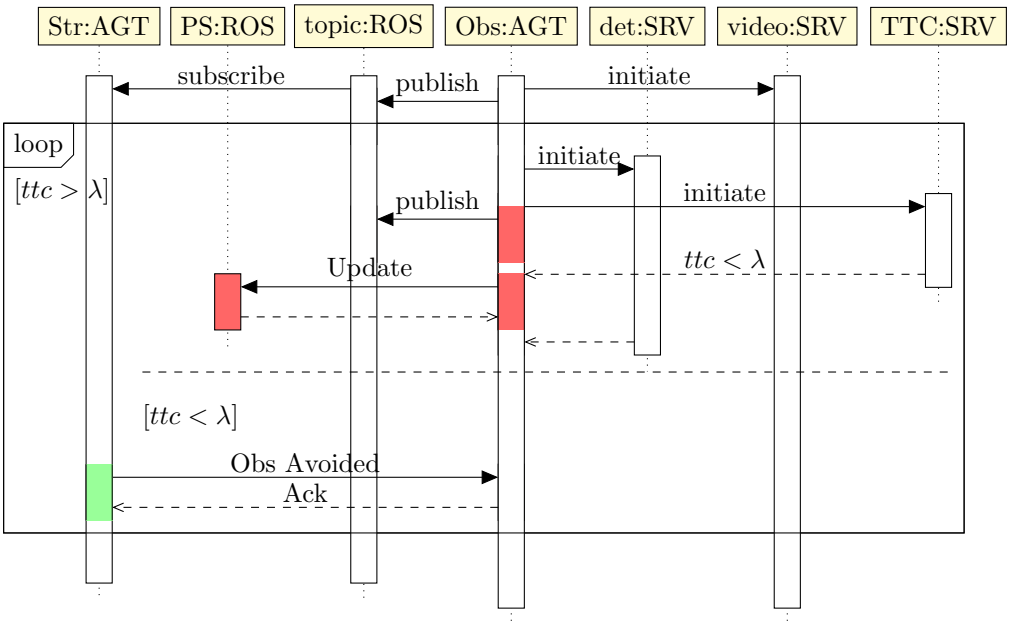


Figure 3.9: Operational sequences for obstacle detection and avoidance. AGT and SRV, represent agent and service. The obstacle agent (Obs:AGT) obtains video stream utilizing video:SRV service and starts detecting obstacles through the obstacle detection service, det:SRV. When an obstacle is detected, Obs:AGT initiates, TTC:SRV to estimate distance to the obstacle. When the estimated distance to the obstacle is less than the critical distance λ, Obs:AGT, updates on the parameter server, PS:ROS. Steering agent (Str:AGT), informs Obs:AGT after the obstacle is avoided.

3.3 Demonstrating Collision Avoidance Behaviour of SOIFRA

This section presents the experiments as well as simulations of the mission explained in the previous section, with SOIFRA. Both simulation and real-time experiments are performed to demonstrate the interoperability and modularity of SOIFRA in accommodating multiple platform independent algorithms. Robot's

mission, as explained above, is to reach a target destination in an unknown environment while avoiding obstacles, utilizing SOIFRA. This mission helps to study the performance of the obstacle agent and the collaboration between steering agent and obstacle agent. Fig.3.10 shows the layout of the simulation environment. Fig. 3.11 and 3.15 shows the operational environments for the simulation and real-time experiments respectively. In Fig. 3.10 white region corresponds to robot operation region and the grey region indicates the target region. A and B (Fig. 3.10) are the robot starting locations. Mission is completed when the robot reaches the target region.

3.3.1 Simulation Experiments Demonstrating Collision Avoidance using SOIFRA

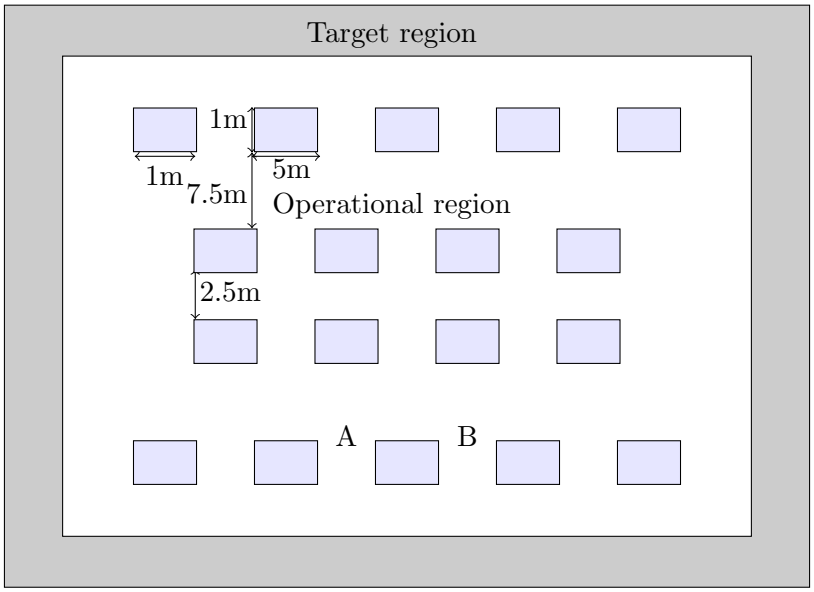


Figure 3.10: Layout of the simulation environment (not drawn to scale). The grey region indicates the target region and the white region indicates the operational region. The mission is completed once the robot reaches the grey target region.

Simulations are carried out utilizing two ground robots (Turtlebot and Clearpath Husky) and two aerial robots (AR Drone and Hector-quadrotor). Details about the robots (Turtlebot, Clearpath Husky, Hector-quadrotor and AR Drone) utilized in simulation experiments are explained in Section. 2.3.2 of Chapter 2. Turtlebot, Clearpath Husky and Hector-quadrotor use simulated

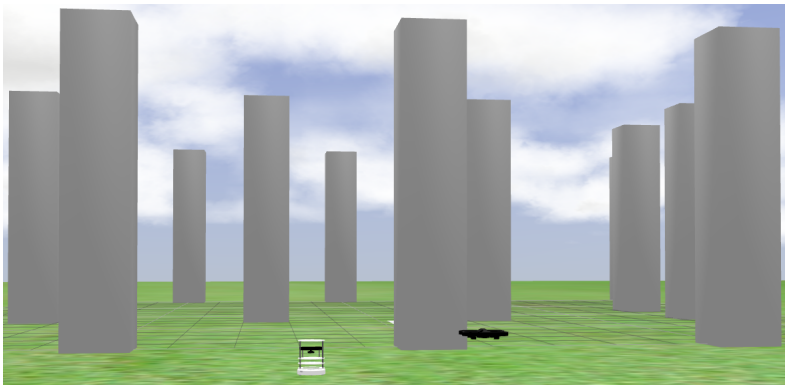


Figure 3.11: Image of operational environment with multiple static obstacles for simulation experiments.

model of Microsoft Kinect as vision sensor, while AR Drone uses a simulated RGB camera. All the cameras produce images with 640x480 resolution. *Gazebo*, a 3D simulator for robots is utilized for simulation. *Gazebo* offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. Open Dynamics Engine (ODE), and open source high performance library for simulating rigid body dynamics is utilized as the physics engine.

Figs. 3.12 and 3.13 show the best results for simulations utilizing TTC-EO and TTC-OF obstacle avoidance algorithms respectively. Each simulation (obstacle avoidance with TTC-OF and TTC-EO) is repeated five times separately. Figs. 3.12 and 3.13 show the X-Y plane (top-view) of the simulation environment. The operating region is a 20x20m square as indicated in Fig. 3.10. The robots start from starting locations A or B. Clearpath Husky and Hector-quadrotor start from A and Turtlebot and AR Drone start from B, utilizing TTC-EO. The positions interchange while TTC-OF is utilized for obstacle avoidance (Clearpath Husky and Hector-quadrotor start from B and Turtlebot and AR Drone start from A). If the robots move out of the operating region (indicated by the grey shade), the mission is complete. The ground robots move with the same velocity and the aerial robots move with the same velocity. The aerial robots move at a higher velocity compared to the ground robots. The robots move forward in a straight line path if there is no obstacle detected or if the distance to the

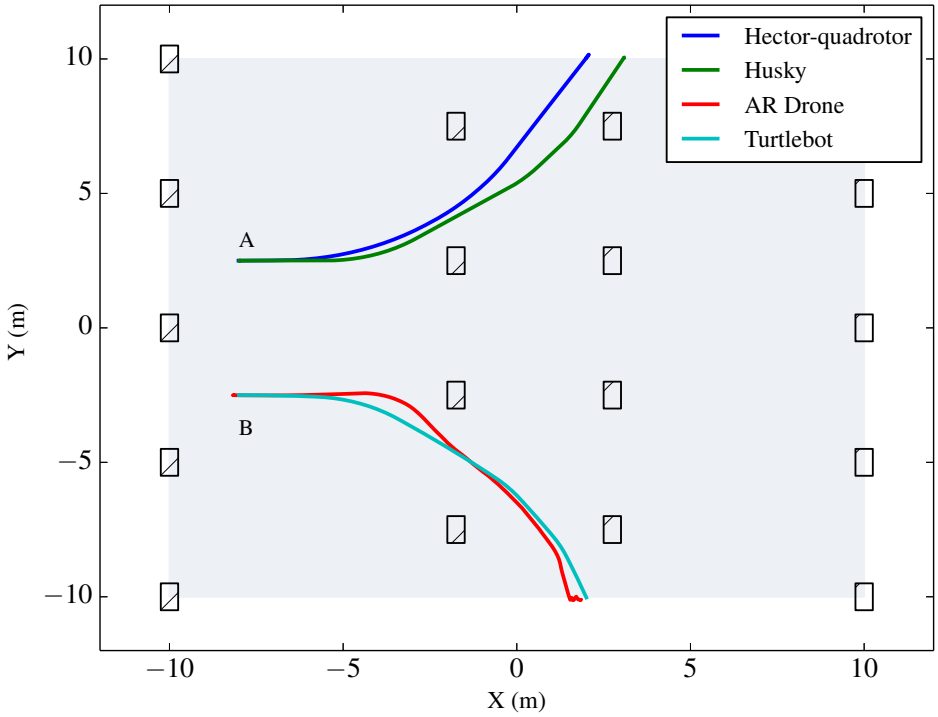


Figure 3.12: Simulation of a mission where two ground robots and two aerial robots use TTC-EO for obstacle avoidance. Top-view (bird's eye view) of the path taken by the robots is shown. Hector-quadrotor and Husky start from A while AR Drone and Turtlebot start from B. Each mission is carried out separately.

obstacle is greater than the critical distance or if the obstacle avoidance process is completed.

Fig. 3.12 shows that all the robots avoid two obstacles before they exit the operational region. There are differences in the path taken by the ground and aerial robots though they follow the same direction. This is due to the inherent differences between a ground robot and an aerial robot. Ground robots are more stable and can turn in a stabilized manner while the aerial robots need some time for stabilization after a turning manoeuvre. Once the distance to the obstacle detected is less than the critical distance the robots perform a turning manoeuvre to avoid the obstacle. If the robot has moved 0.8m from the point where a turning manoeuvre is initiated and if there is no obstacle detected, the robot stops the turning manoeuvre and moves forward in the same direction. The direction of the turn depends on the position of the obstacle in the robot's path. If the robot senses that obstacle is located to its left, a right turn is performed and vice-versa.

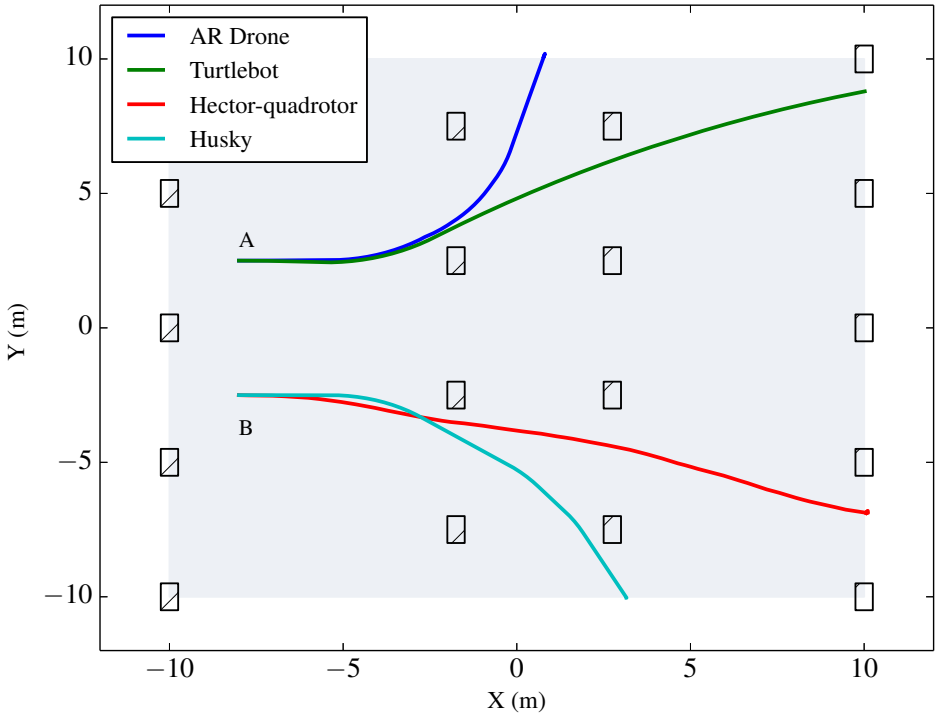


Figure 3.13: Simulation of a mission where two ground robots and two aerial robots use TTC-OF for obstacle avoidance. Top-view (bird's eye view) of the path taken by the robots is shown. AR Drone and Turtlebot start from A while Hector-quadrotor and Husky start from B. Each mission is carried out separately.

This is the reason for different turning directions when the robots start from A and B, though the operational environment is symmetrical.

Figs. 3.13 shows the simulation results when TTC-OF is utilized for obstacle avoidance. Turtlebot and AR Drone start from A and Clearpath Husky and Hector-quadrotor start from B. Figs. 3.13 shows that Clearpath Husky and AR Drone avoid two obstacles while Turtlebot and Hector-quadrotor avoid only one obstacle before exiting the operating region. This is due performance differences between TTC-OF and TTC-EO in estimating the distance to the obstacle. TTC-OF tends to have a higher error than TTC-EO and it can lead to early obstacle avoidance, as indicated by the path traveled by Turtlebot and Hector-quadrotor. Since the obstacle avoidance is initiated early, the robots complete their turning manoeuvre and start moving straight early. This is the reason for different paths taken by the Turtlebot and Hector-quadrotor.

3.3.2 Real-time Experiments Demonstrating Collision Avoidance with SOIFRA

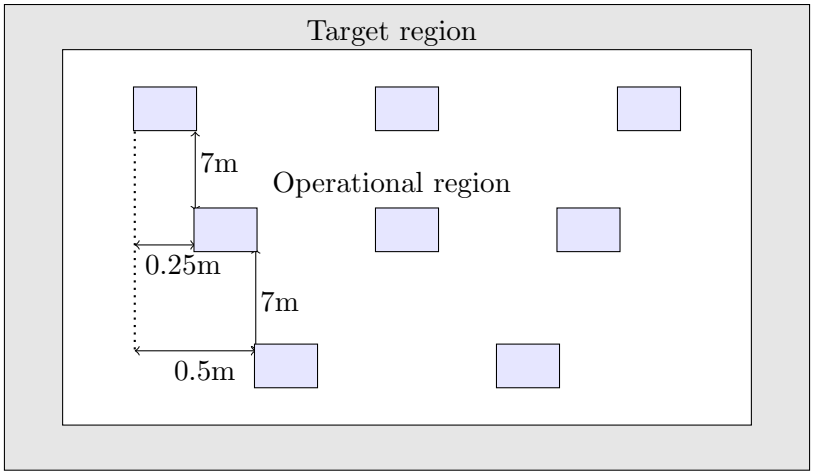
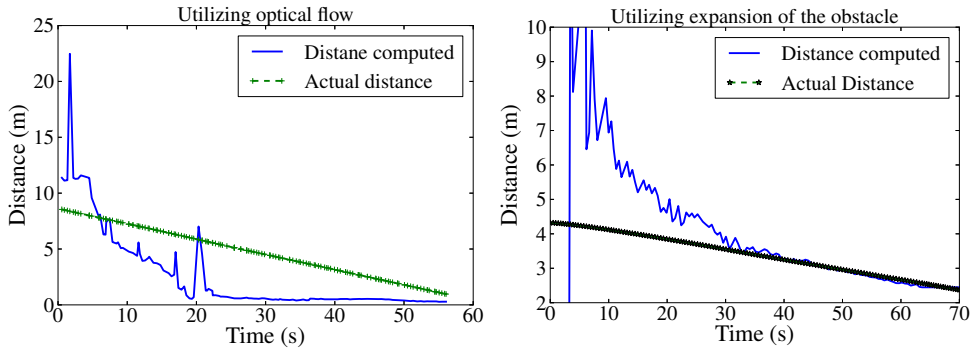


Figure 3.14: Layout of the environment for real-time experiments (not drawn to scale). The grey region indicates the target region and the white region indicates the operational region. The mission is completed once the robot reaches the grey target region.

Figure 3.15: Image of real-time operational environments with multiple static obstacles

Fig. 3.14 shows the layout of the environment for the real time experiments. The operational environment is 4m wide, and 14m long. Turtlebot and AR Drone are used for real-time experiments. Details and specifications of Turtlebot and AR Drone utilized for experiments are discussed in Section. 2.3.3 of Chapter 2. Turtlebot uses a Microsoft Kinect as vision sensor while AR Drone utilizes its on-board camera. */odom* topic of Turtlebot (combination of wheel odometry and IMU) is utilized for Tutlebot position estimation while localization of AR



(a) Estimated utilizing optical-flow-based time-to-contact algorithm (3.4)

(b) Estimated utilizing expansion-of-obstacle-based algorithm (3.7

Figure 3.16: Comparison between distance to the obstacle estimated utilizing TTC-OF and TTC-EO

Table 3.5: Comparison among mean error, mean absolute error and mean squared error of distance to the obstacle, computed utilizing TTC-OF and TTC-EO

True distance	TTC-OF (m)			TTC-EO (m)		
to the obstacle(m)	ME	MAE	MSE(m^2)	ME	MAE	MSE(m^2)
4.5 - 4.0	-3.762	3.762	14.170	0.780	0.800	2.451
4.0 - 3.5	-3.255	3.255	10.625	0.801	0.801	0.797
3.5 - 3.0	-2.717	2.717	7.401	0.066	0.080	0.0104
3.0 - 2.5	-2.194	2.194	4.839	-0.073	0.073	0.063

Drone is based on its */ardrone/odometry* topic (IMU). Figs. 3.17 and 3.18 show the results for real-time experiments utilizing TTC-EO and TTC-OF obstacle avoidance algorithms respectively. Both the robots start from the same starting location and the starting location is changed collectively when obstacle avoidance algorithm is changed. Turtlebot and AR Drone travel with different velocities, velocity of AR Drone being higher. The experiments (obstacle avoidance using TTC-OF and TTC-EO) are repeated three times separately.

Fig.3.17 shows the best experimental results when TTC-EO is utilized for obstacle avoidance. Both Turtlebot and AR Drone start from start location (0,0) and move forward (locations are expressed as Cartesian co-ordinates). Once the distance to the obstacle estimated is less than the critical distance, robots undergo turning manoeuvre (obstacle avoidance). It can be seen from Fig.3.17 that both the robots avoid two obstacles before exiting the operation region. But AR Drone and Turtlebot travel in different directions while avoiding the first obstacle as

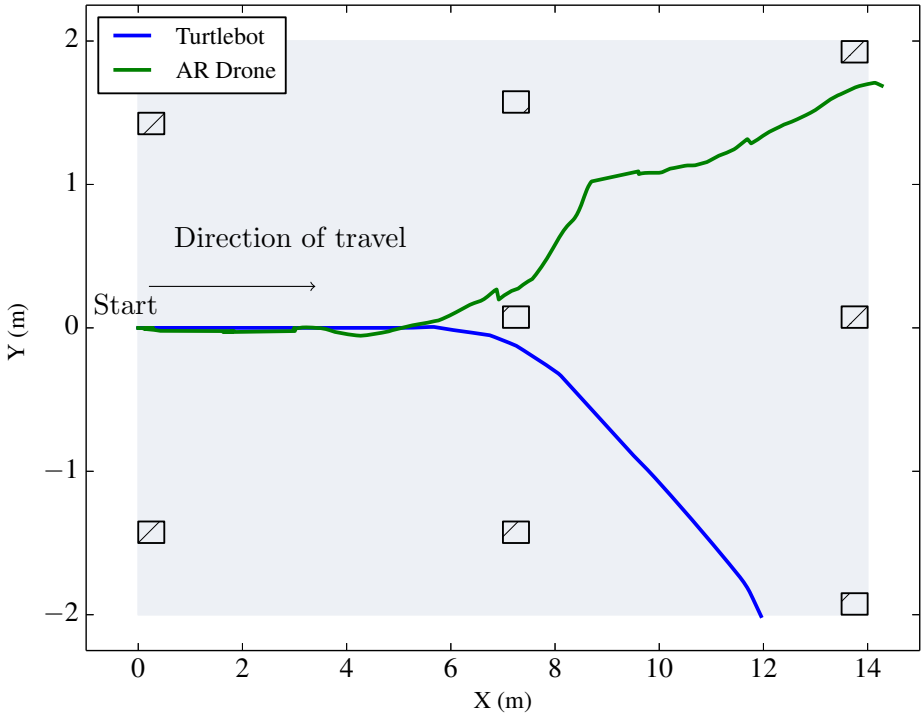


Figure 3.17: Top-view (bird's eye view) of the path taken by AR Drone and Turtlebot while completing the mission in real-time. TTC-EO is utilized for obstacle avoidance. Each mission is carried out separately.

the location of the obstacle detected is different. Non-linear movement of AR Drone at the final stage of obstacle detection process results in different turning direction for the AR Drone. Fig.3.18 shows the best experimental results when TTC-OF is utilized for obstacle avoidance. Turtlebot and AR Drone start from Start location (14,1.75) and move towards negative X-axis. AR Drone avoids two obstacles one at (7,0) and the other at (-1.5,-2), while the turtlebot avoids one obstacle at (7,0), before exiting the operational region. Both the robots have the same turning direction but the obstacle avoidance process is initiated earlier for the turtlebot. Fig. 3.16 shows the distance to the obstacle estimated utilizing TTC-OF (Fig. 3.16a) and TTC-EO (Fig. 3.16b). It can be seen that distance to the obstacle estimated utilizing TTC-EO follows the true distance after 4.5m, while the distance to the obstacle estimated utilizing TTC-OF decreases closer to 3m. Table. 3.5 shows the mean-error (ME), mean-absolute-error (MAE) and mean-squared-error (MSE) performances of TTC-EO and TTC-OF averaged over the three separate runs. In order to study the relation between the performance

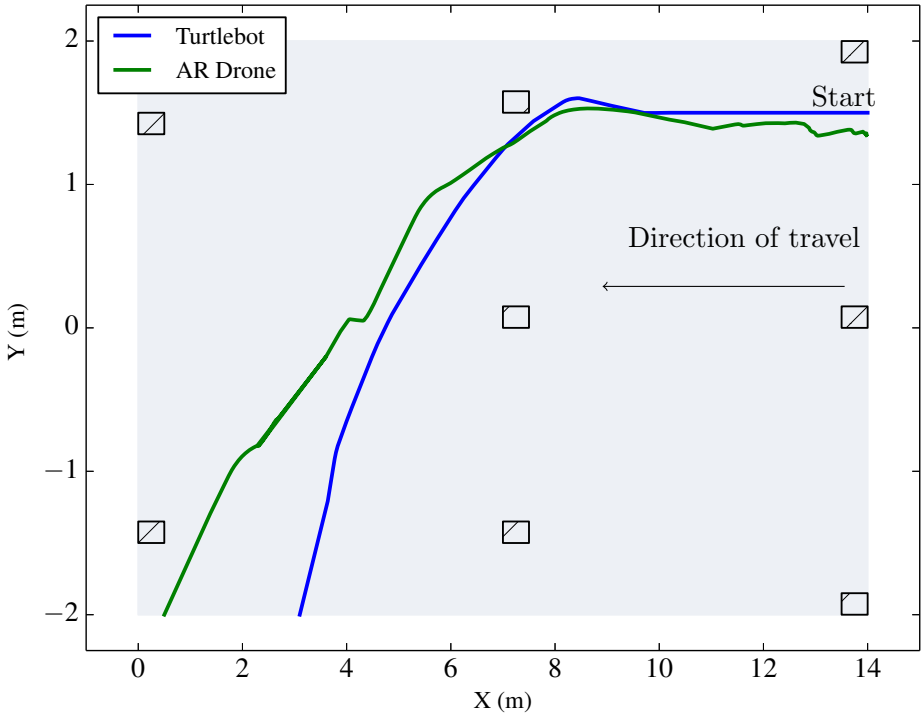


Figure 3.18: Top-view (bird's eye view) of the path taken by AR Drone and Turtlebot while completing the mission in real-time. TTC-OF is utilized for obstacle avoidance. Each mission is carried out separately.

of TTC-EO and TTC-OF with respect to the distance to the obstacle, the error performances are analysed in ranges of 0.5m. The error performance measures are calculated using the following,

$$ME = \frac{1}{n}\sum_{i=1}^{n}(\hat{d} - d_{true}) \tag{3.15}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|(\hat{d} - d_{true})| \tag{3.16}$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{d} - d_{true})^2 \tag{3.17}$$

where, n is the number of samples in the interval, $\hat{d}$ is the distance to the obstacle estimated and d_{true} is the true distance to the obstacle. It can be seen from the error measures (Table. 3.5) that TTC-EO performs better compared to TTC-OF. This is evident from the simulation and experimental results (Figs. 3.12, 3.13, 3.17 and 3.18). The error in distance to the obstacle estimated utilizing TTC-OF decreases as the robot moves closer to the obstacle as explained

in Section. 3.1.2.4. It is noted that the ability of the canny contours algorithm to detect the edges of the obstacle varies with respect to the lighting conditions (inherent problem with most of the vision based algorithms). The low and high thresholds of the canny contour algorithm needs to be changed depending on day/night lighting conditions.

In both simulation and real-time experiments, only the time-to-contact service is modified. The rest of the services and the framework are not affected by this change. Similarly, the obstacle detection service may also be modified without affecting the rest of the framework. This demonstrates the modularity of the SOIFRA framework designed. Successful utilization of SOIFRA for collision avoidance on different platform such as aerial vehicle (AR Drone and Hector-quadrotor) and ground vehicle (Turtlebot and Clearpath Husky) also strengthens the claim that SOIFRA interoperable.

3.4 Summary

This chapter discusses about the incorporation of collision avoidance behaviour into SOIFRA. The main focus of SOIFRA framework is to generalize platform independent algorithms for unmanned aerial and ground robots. To demonstrate this, two obstacle avoidance algorithms (TTC-OF and TTC-EO) are utilized on aerial as well as ground robots. Safe operating range for utilizing TTC-OF and TTC-EO is identified as 5m through series of experiments. The behaviour based nature of SOIFRA allows the agents to dynamically update their knowledge in real-time leading to effective collision avoidance. Service oriented nature of SOIFRA helps to achieve this modularity whereby a service can be replaced with other similar services without affecting other components of the framework. Though the control mechanisms for aerial robots and ground robots are completely different, the algorithms and mechanisms for obstacle detection and obstacle avoidance are generalized. SOIFRA utilizes this concept to achieve interoperability across diverse robotic platforms such as aerial and ground robots. Interoperability of SOIFRA is established by utilizing the framework for completing the same

mission on four different robotic platforms (Turtlebot, Clearpath Husky, AR Drone and Hector-quadrotor) for simulations and two diverse robotic platforms (Turtlebot and AR Drone) for real-time experiments. Two agents, a steering agent, for navigating the robot and obstacle agent, for obstacle detection and avoidance utilized are successful in completing the missions.

Chapter 4

Real-time Path Generation and Tracking with SOIFRA

"The truest wisdom is a resolute determination"

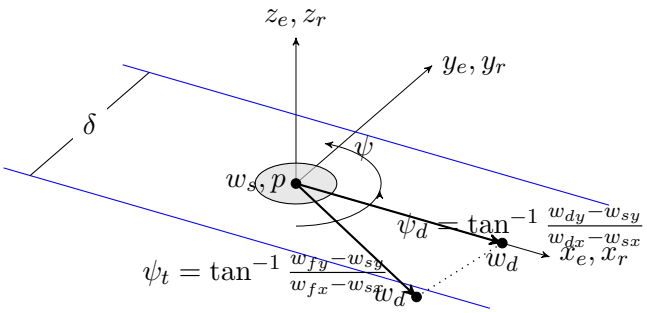
– Napolean Bonaparte

Navigation strategies and robot guidance algorithms are essential for successful autonomous operation of unmanned robots. Navigation is a process or activity of accurately estimating a robot's position, and planning and directing the robot towards the planned path. Navigation is essential for autonomous mobile robots as most of the autonomous robot missions require the robots to travel from one location to the other safely, without any collision or without getting lost. Localization, path-planning and path-tracking are the three essential components of navigation. Localization is the ability of a robot to establish its own position and orientation with respect to a frame of reference. Path-planning enables the robot to select and identify a suitable path for it to traverse in the environment [114]. Path-tracking algorithms determine the commanded heading angle for the robot that will enable it to accurately trace a given path [125].

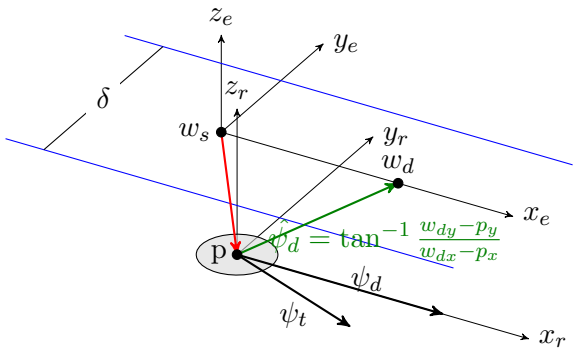
Robot guidance algorithms generally provide path-planning and path-tracking functionalities to autonomous robots. This chapter mainly focuses on adding path-generation and path-tracking functionalities into SOIFRA. A platform

independent, geometry-based real-time path-tracking algorithm is developed and incorporated in SOIFRA.

4.1 Vector Directed Path-Generation and Tracking (VDPGT)



(a) Scenario where a robot is within transition boundaries.



(b) Scenario where a robot crosses the transition boundary. $p = w_s$ and $\psi_d = \hat{\psi}_d$, once the robot crosses a transition boundary.

Figure 4.1: Illustration of the parameters used in VDPGT. The shaded circle represents a robot.

Robot guidance algorithms generate path(s) from a robot's current location to its destination and determine the changes required in robot's velocity, rotation and acceleration in order to follow the path generated. The VDPGT algorithm developed is a robot guidance algorithm, comprising of path-generation and path-following components. VDPGT is designed to guide robots to their destinations in the shortest path possible. Path-generation is the action of determining the path a robot has to follow, in order to successfully reach a destination. Path-following is the action of determining a robot's heading angle that leads to accurate path-

tracking. VDPGT provides path-generation and path-following abilities for both aerial and ground robots. Ground robots move in a two dimensional X-Y plane while aerial robots move in a three dimensional X-Y-Z plane. Without loss of generality, an aerial vehicle can be considered to be moving in a two dimensional plane while there is a change in the robot's altitude and in conditions where there is no change in the robot's altitude. To ensure interoperability across unmanned aerial and ground vehicles, VDPGT achieves path-following in a two dimensional plane (X-Y plane for a ground robot and X-Y or X-Z or Y-Z planes for aerial vehicles).

Let x_e, y_e, z_e and x_r, y_r, z_r represent earth and robot co-ordinate frames respectively (Figs. 4.1a and 4.1b). VDPGT requires initial location of the robot w_s, the destination to be reached by the robot w_d, current position of the robot p and orientation of the robot in Z plane (yaw of the robot) ψ_t for path-following. A straight line connecting the source and destination is the shortest path between them. VDPGT utilizes this concept for path-generation. A straight line between w_s and w_d is the desired trajectory to follow. The desired orientation of the robot ψ_d, is the angle between $\overrightarrow{w_s}$ and $\overrightarrow{w_d}$. The minimum-acceptable-orientation-difference ϵ_ψ, is the minimum acceptable absolute difference between desired orientation (ψ_d) and actual orientation (ψ_t) of the robot. Transition boundaries are defined to ensure successful path-following. The transition boundaries are defined parallel to the desired path, with w_s as its mid-point. A robot is considered to be away from its path if it crosses the transition boundaries defined. δ represents the distance between the transition boundaries defined. ϵ_d is the minimum acceptable error between destination reached and destination to be reached by the robot.

Algorithm 4.1 shows the pseudo code of VDPGT algorithm developed. VDPGT is made up of two loop structures. The inner loop controls the orientation and the outer loop controls the position of the robot. The inner loop ensures that the orientation of the robot is less than or equal to the minimum-acceptable-orientation-difference ($\psi_d - \psi_t \leq \epsilon_\psi$). The outer loop moves the robot forward, if the robot's destination is not reached. As shown in Fig. 4.1b, if the robot crosses

Algorithm 4.1 Vector Directed Path Generation and Tracking Algorithm

Input: initial location (w_{sx}, w_{sy}), destination (w_{dx}, w_{dy}), current location (p_x, p_y), target heading ψ_d, current heading ψ_t, minimum-acceptable-orientation-difference ϵ_ψ, distance between the transition boundaries δ such that $\delta/2$ is less than minimum acceptable error in destination reached and to be reached by the robot (ϵ_d).

$\psi_d = \tan^{-1}(\frac{w_{dy}-w_{sy}}{w_{dx}-w_{sx}})$ (Calculate the desired orientation)
$p^* = \frac{(p-w_s)^T(w_d-w_s)}{\|w_d-w_s\|^2}$ (Calculate the position of the robot along the path $p^* \in [0,1]$)
while $p^* < 1$ **do**
  if $\psi_d - \psi_t \leq \epsilon_\psi$ **then**
    Move forward
    if $\mid p_y - w_{1y} \mid > \frac{\delta}{2}$ **then**
      $w_s = p$
      update p^*
      $\hat{\psi}_d = \tan(\frac{w_{dy}-w_{sy}}{w_{dx}-w_{sx}})$
      $\psi_d = \hat{\psi}_d$
    end if
  else if $(\psi_d - \psi_t) > 0$ **then**
    Turn Left
  else
    Turn Right
  end if
end while

a transition boundary, initial position of the robot w_s is reinitialized leading to the current position of the robot p being assigned as robot's initial location $(w_s = p)$ and new desired orientation $\hat{\psi}_d$, based on new w_s, being calculated. This enables the robot to successfully move towards its destination even in the presence of disturbances. However for the robot to successfully reach its desired destination ϵ_ψ must be small. Proposition 1 explains the reason for the small value of parameter ϵ_ψ and the condition $\frac{\delta}{2} < \epsilon_d$, for a robot to reach its destination. Proposition 1 also establishes that the error in final destination reached by the robot is always bounded (less than ϵ_d).

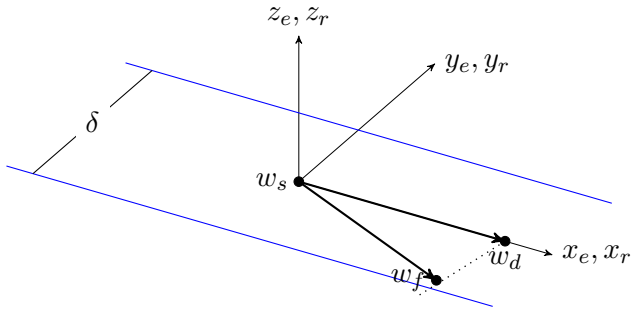


Figure 4.2: Illustration of the parameters used in Proposition 1.

Proposition 1. *In Fig. 4.2, if w_s is starting position of the robot (or the starting position of the robot after a reset if the robot crosses the transition boundary), w_d is the desired destination and w_f is the final destination reached by the robot utilizing Algorithm 4.1, then when Algorithm 4.1 terminates, $||w_f - w_d|| < \epsilon_d$, where ϵ_d is the minimum acceptable error in destination reached and to be reached by the robot.*

Proof. Algorithm 4.1 terminates when $p^* = 1$.

$$p^* = \frac{(p - w_s)^T (w_d - w_s)}{\| w_d - w_s \|^2}, \tag{4.1}$$

where p is the current position of the robot.
If w_f is the final position reached by the robot, then

$$p^* = \frac{(w_f - w_s)^T (w_d - w_s)}{\| w_d - w_s \|^2} = 1 \tag{4.2}$$

$$(w_f - w_s)^T (w_d - w_s) = \| w_d - w_s \|^2 \tag{4.3}$$

$$(w_f - w_s)^T (w_d - w_s) = (w_d - w_s)^T (w_d - w_s) \tag{4.4}$$

$$(w_f - w_s - w_d + w_d)^T (w_d - w_s) = (w_d - w_s)^T (w_d - w_s) \tag{4.5}$$

$$((w_f - w_d) + (w_d - w_s))^T (w_d - w_s) = (w_d - w_s)^T (w_d - w_s) \tag{4.6}$$

$$((w_f - w_d)^T + (w_d - w_s)^T)(w_d - w_s) = (w_d - w_s)^T (w_d - w_s) \tag{4.7}$$

$$(w_f - w_d)^T (w_d - w_s) + (w_d - w_s)^T (w_d - w_s) = (w_d - w_s)^T (w_d - w_s) \tag{4.8}$$

$$(w_f - w_d)^T (w_d - w_s) = 0 \tag{4.9}$$

Eqn. 4.9 implies that $\angle w_s w_d w_f = 90^0$ when Algorithm 4.1 terminates. That is $||w_f - w_d||$ is the perpendicular distance from w_f to w_d. But Algorithm 4.1 ensures that w_s is reset when the perpendicular distance is more than $\frac{\delta}{2}$ (line 6 in Algorithm 4.1). Hence $||w_f - w_d|| < \epsilon_d$ at $p^* = 1$. □

Hence if $\frac{\delta}{2} < \epsilon_d$, the robot will reach its destination with minimum error in position (small $||w_d - w_f||_2$). VDPGT is a simple real-time path-generation

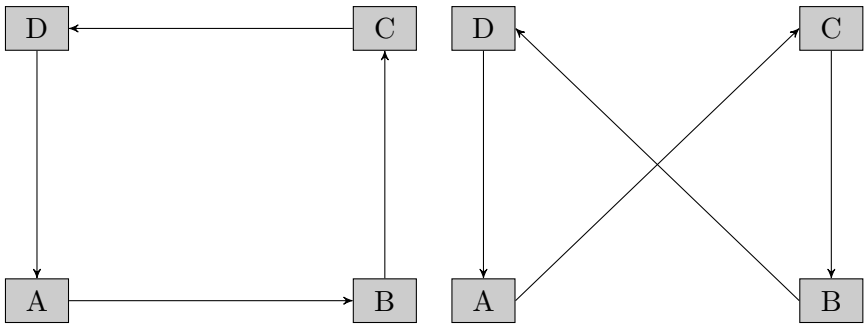
and path-following algorithm. VDPGT is most useful in scenarios where the next waypoint is unknown (in most of the exploration or search missions). The simulation and experimental studies with ground and aerial robots, discussed in sub-sections 4.1.1 and 4.1.2, demonstrate the performance of VDPGT.

4.1.1 Performance Evaluation of VDPGT through Simulation Experiments

A Turtlebot, ClearPath Husky, Hector-quadrotor and AR Drone are utilized for the simulation studies. All the four robots are capable of making 360^0 turns with zero turning radius. Details about the robots (Turtlebot, Clearpath Husky, Hector-quadrotor and AR Drone) used in the simulation experiments are explained in Section 2.3.2 of Chapter 2. The robots are made to visit four locations (waypoints) in different sequences utilizing VDPGT for path-generation and path-following. Fig. 4.3 shows the layout of the waypoints for the simulation experiments. Only information about the current waypoint is made known to the robot. The next waypoint is made known only when a robot reaches a current waypoint. Two cases of path-following sequences are studied. In both the cases the robots start from location A and end at location A. For Case – 1 the robots visit locations A, B, C, D and go back to A while for Case – 2, robots visit locations A, C, B, D and go back to A. The optimal path for Case – 1 and Case – 2 are shown in Figs. 4.3a and 4.3b. For the simulation experiments the minimum acceptable orientation difference is set at 3^0, the minimum difference in destination reached and to be reached by the robot ϵ_d and the distance between the transition boundaries δ are set as 2m. Since VDPGT considers only two coordinates (source and destination), each waypoint is considered as a destination. Source and destination coordinates are re-initialized once the robots reach a waypoint and desired orientation ψ_d is calculated accordingly.

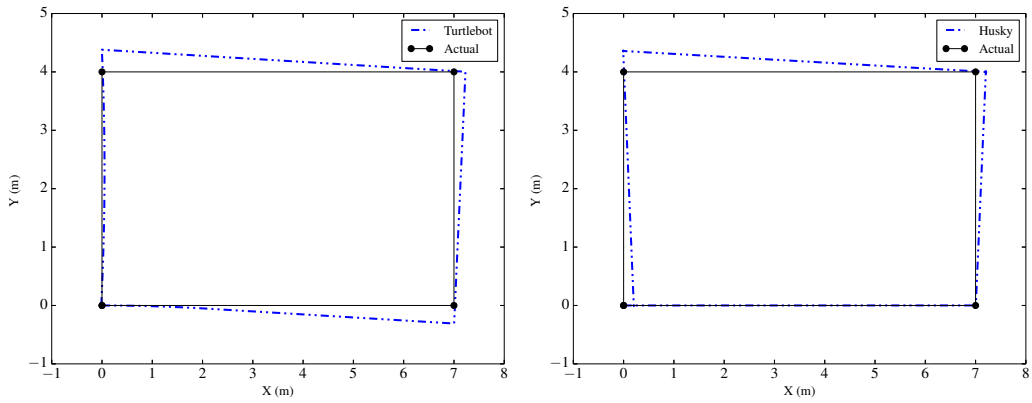
4.1.1.1 Case – 1

To complete the task for Case – 1, the robots go through A (0,0) - B (7,0) - C (7,4) - D (0,4) - A (0,0) as shown in Fig.4.3a. The robots should make a 90^0 turn after



(a) Case – 1 (b) Case – 2

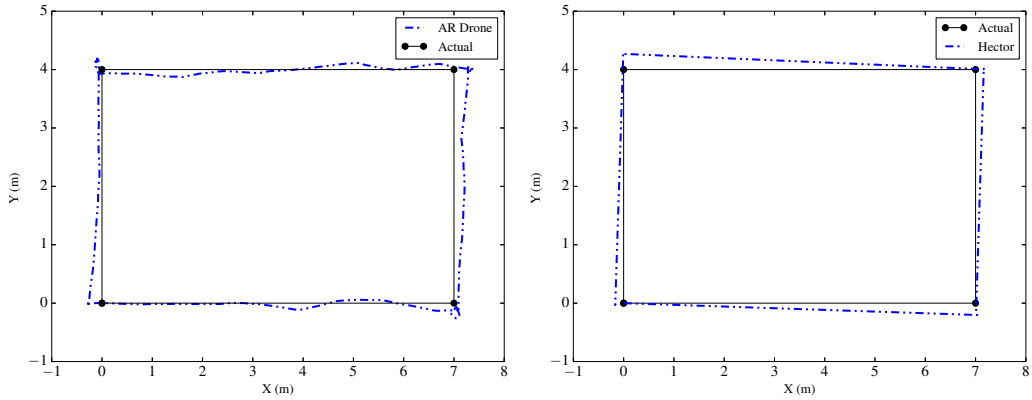
Figure 4.3: Sequences of path-following utilized for simulation and real-time experiments. The robots have to start from A and end at A following the direction specified.



(a) Turtlebot (b) ClearPath Husky

Figure 4.4: Simulation results for Case – 1 when ground robots are utilized

reaching each waypoint. All the four robots utilized for the simulations have zero turning radius. Fig.4.5, Figs. 4.4a, 4.4b, 4.5a and, 4.5b show the path travelled by Turtlebot, Clearpath Husky, AR Drone and Hector-quadrotor respectively. The solid line in the above mentioned figures indicate the shortest path through the waypoints. Figs. 4.4a, 4.4b, 4.5a and, 4.5b show that the error in distance reached and distance to be reached by the robots in each leg of travel (A-B, B-C, C-D and D-A) is less than ϵ_d (2m). It can be seen that all the robots follow the optimal path closely and the cross-track error is in the order of centimeters.



(a) AR Drone (b) Hector-quadrotor

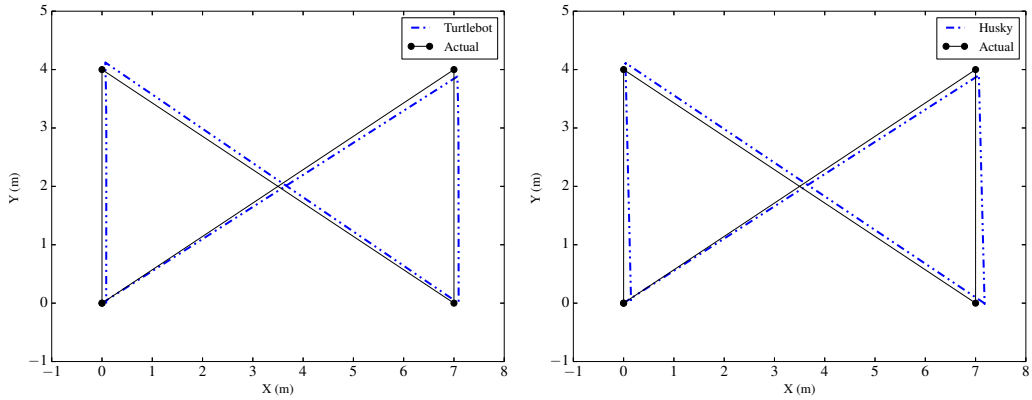
Figure 4.5: Simulation results for Case – 1 when aerial robots are utilized

4.1.1.2 Case – 2

To complete the task for Case – 2, the robots go through A (0,0) - C (7,4) - B (7,0) - D (0,4) - A (0,0) as shown in Fig.4.3b. The robots are positioned at A facing B (yaw angle is 0^0 when robot faces B) at the start of all the simulation experiments. The robots have to turn approximately -30^0 to reach C. After reaching C the robots have to make a 120^0 turn to reach B. The robots then have to make a 30^0 and -30^0 turns to reach D from B and A from D respectively. The path travelled by Turtlebot, Clearpath Husky, AR Drone and Hector-quadrotor are shown in Figs. 4.6a, 4.6b, 4.7a and, 4.7b respectively and the solid line in these figures represent the shortest path through the waypoints. It can be seen from the path travelled by the robots that all the robots are able to successfully follow the shortest path through the waypoints utilizing VDPGT.

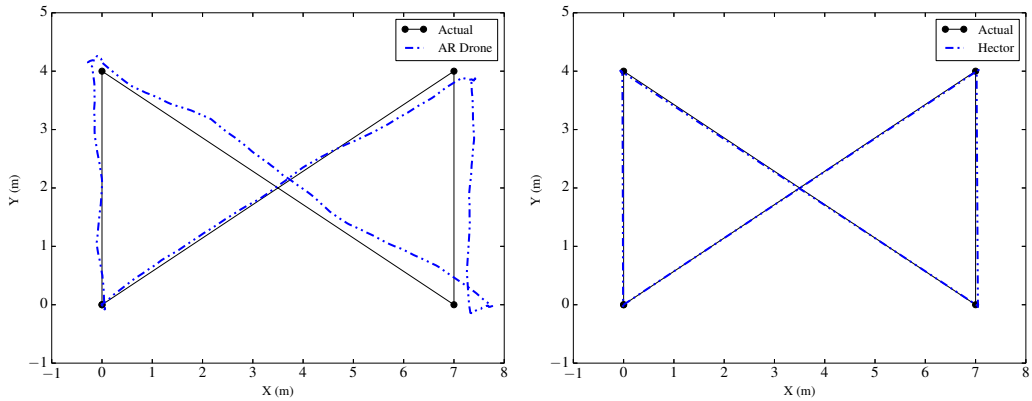
4.1.2 Performance Evaluation of VDPGT through Real-time Experiments

Performance of VDPGT in real-time is studied by testing the algorithm on a Turtlebot and an AR Drone. Details and specifications of Turtlebot and AR Drone are elaborated in Section 2.3.3 of Chapter 2. Both the robots are subjected to Case – 1 and Case – 2 discussed in Section 4.1.1 and are directed to reach four destinations A(0,0), B(7,0), C(7,4) and D(0,4) utilizing VDPGT. In order



(a) Turtlebot (b) ClearPath Husky

Figure 4.6: Simulation results for Case – 2 when ground robots are utilized



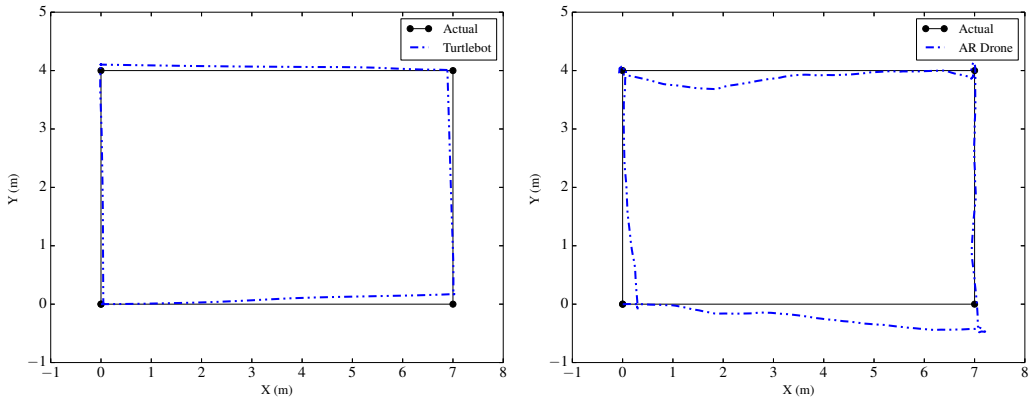
(a) AR Drone (b) Hector-quadrotor

Figure 4.7: Simulation results for Case – 2 when aerial robots are utilized

to create a real-time path exploration scenario, robot's next destination is known only when the robot reaches its current destination. Figs. 4.3a and 4.3b shows the sequences of path-following for the two cases of the real-time exploration. In both the cases the robots start from location A and end at location A. For Case – 1 the robots visit locations A, B, C, D and go back to A while for Case – 2, robots visit locations A, C, B, D and go back to A. For all the experiments the minimum acceptable orientation difference is set at 3^0, the minimum difference in destination reached and to be reached by the robot ϵ_d and the distance between the transition boundaries δ are set as 2m. The performance of VDPGT algorithm is almost identical in both simulation and real-time experiments. VDPGT considers only two coordinates (source and destination), each waypoint is considered as a

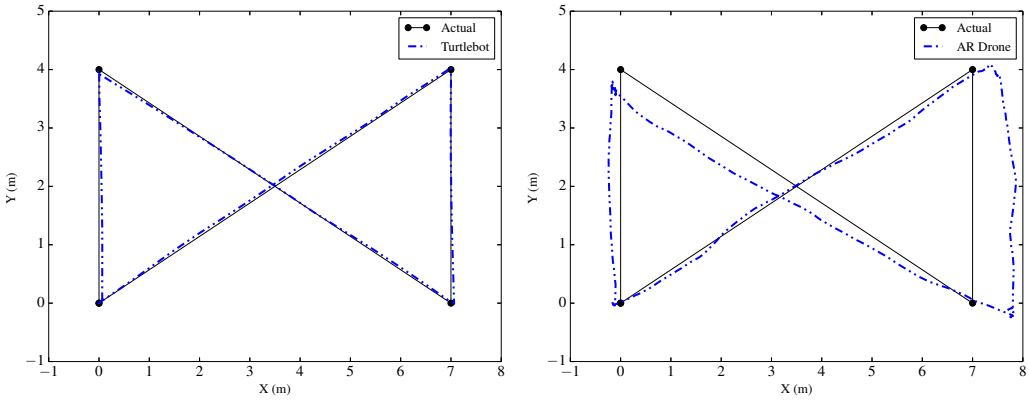
destination. Source and destination coordinates are re-initialized once the robots reach a waypoint and desired orientation ψ_d is calculated accordingly. Figs. 4.8 and 4.9 show the experimental results for Case – 1 and Case – 2 respectively.

It can be seen that both the Turtlebot and AR Drone follow the shortest path (indicated by solid lines) and reach the destinations with minimal error. However it is evident that the path travelled by a Turtlebot is smooth compared to the path travelled by AR Drone. This difference is mainly due to the nature of the two robots as ground robots are more stable and easier to control compared to aerial robots. Even-though the path travelled by the aerial robot is less smooth, Figs.4.8b and 4.9b show that the aerial robot follows the shortest path closely with cross-track error in the order of centimeters. Figs. 4.10, 4.11, 4.12 and 4.13 show the deviation plots for Turtlebot and AR Drone while performing Case – 1 and Case – 2 respectively. In each of these plots the subplots show the deviation from the ideal path, as the robots move from one way-point to the other. The deviation plots show that the maximum deviation obtained is around 0.17m for Turtlebot and 0.8m for AR Drone (including both the cases of real-time experiments). These deviation plots confirm that VDPGT algorithm can generate and follow a trajectory with less than 1m error. These results demonstrate that VDPGT can successfully be utilized on both aerial and ground robots for path-generation and path-tracking.



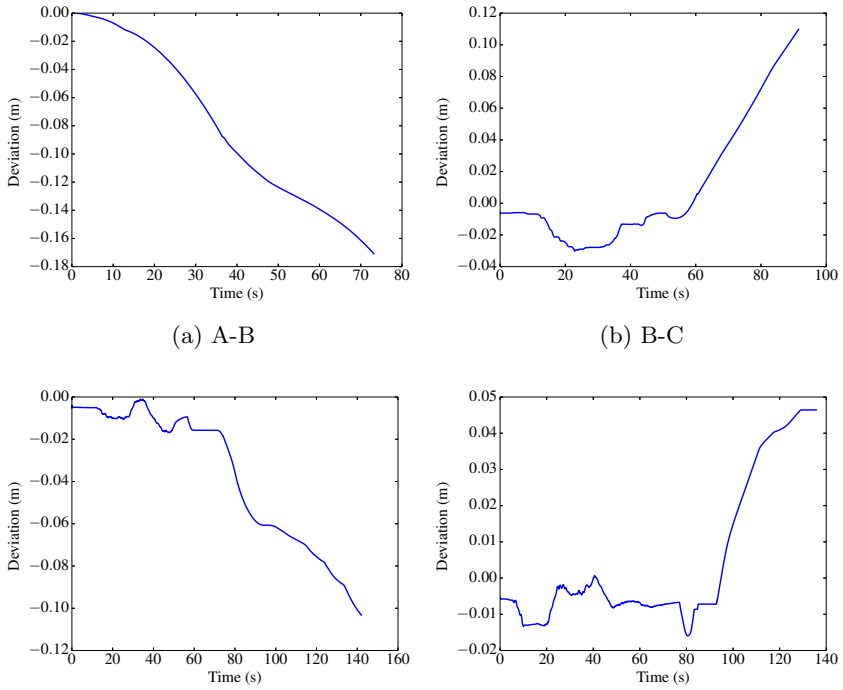
(a) Turtlebot (b) AR Drone

Figure 4.8: Real-time experiment results for Case – 1



(a) Turtlebot (b) AR Drone

Figure 4.9: Real-time experiment results for Case – 2

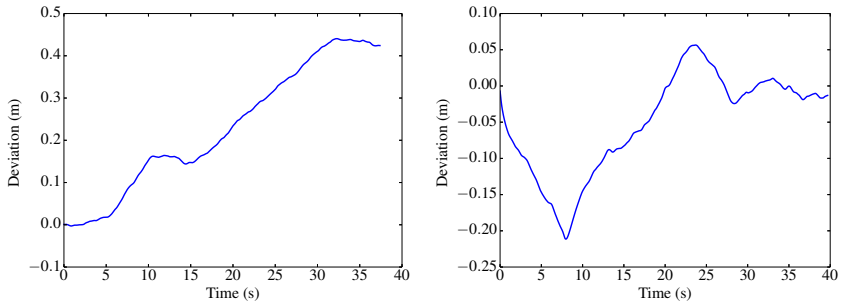


(a) A-B (b) B-C

(c) C-D (d) D-A

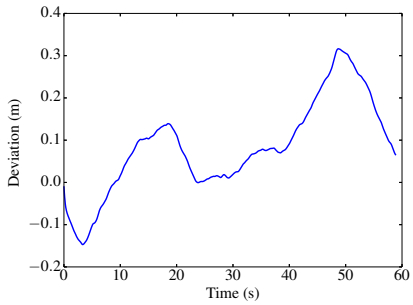
Figure 4.10: Deviation from the actual path calculated for real-time experiments on Turtlebot for Case – 1. The subplots show the deviation while the robot moves through different waypoints.

Both Turtlebot and AR Drone have zero minimum turning radius (tr). Both robots are able to make 90^0 turns with less translation (change in x-y or x-y-z coordinates). However there are many robots that do not have zero turning radius. If the minimum turning radius of the robot is greater than $\delta/2$, the robot will be outside the transition boundary at the end of the orientation control

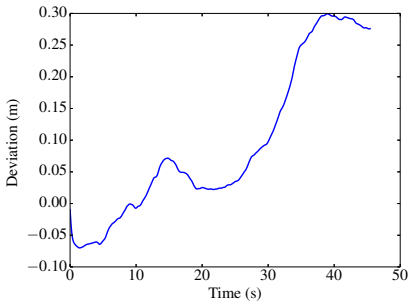


(a) A-B

(b) B-C



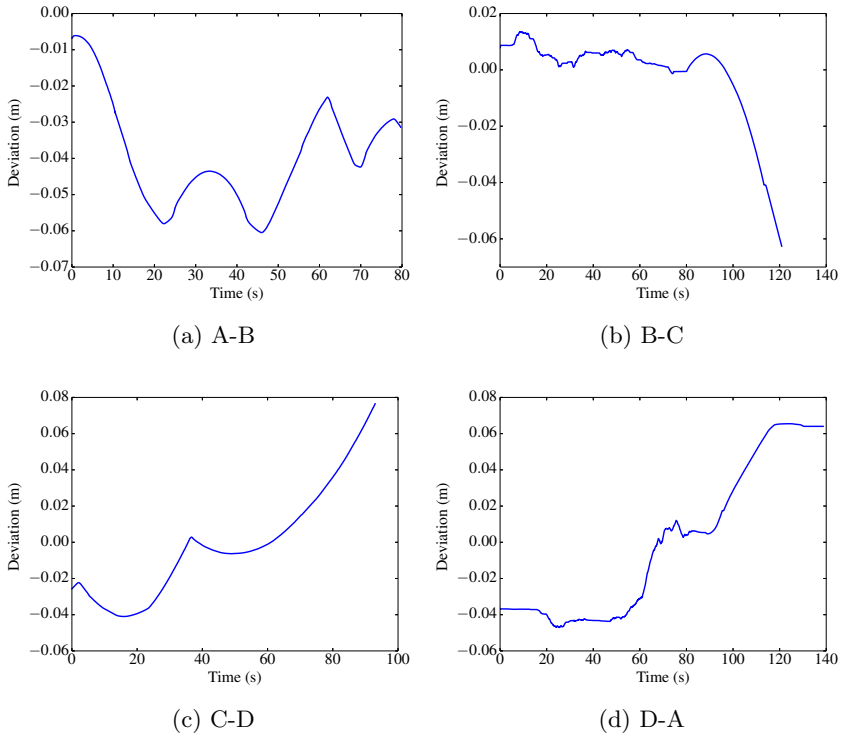
(c) C-D



(d) D-A

Figure 4.11: Deviation from the actual path calculated for real-time experiments on AR Drone for Case – 1. The subplots show the deviation while the robot moves through different waypoints.

loop. VDPGT then re-initializes the current location of the robot and continues with the orientation control loop instead of moving to the translation control loop. Fig. 4.14 shows the path-taken by robots with different turning radius while performing 90^0 turn, utilizing VDPGT. It can be seen that a for a robot with $tr < \delta/2$, VDPGT proceeds to the transition control loop immediately after completing the orientation control loop, while for robots with $tr > \delta/2$, VDPGT continues with the orientation control loop until the robot is within the transition boundaries and $\psi_d - \psi_t \leq \epsilon$. It can also be seen that, greater the tr of a robot, larger the time taken to complete the orientation control loop.



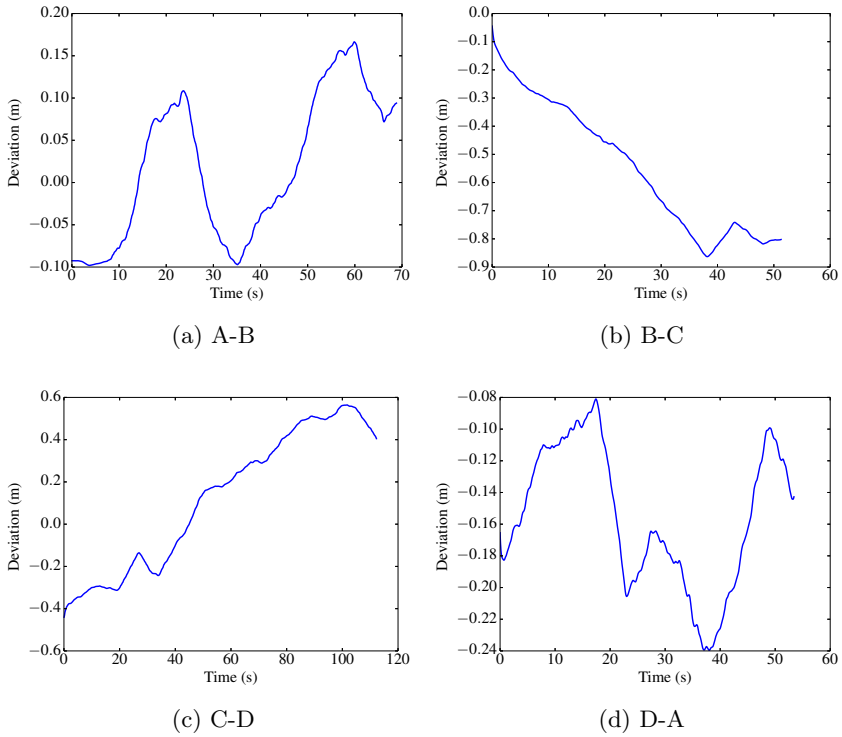
(a) A-B (b) B-C

(c) C-D (d) D-A

Figure 4.12: Deviation from the actual path calculated for real-time experiments on Turtlebot for Case – 2. The subplots show the deviation while the robot moves through different waypoints.

4.2 Coordinated Vision-based Localization for Unmanned Aerial and Ground Vehicles Utilizing VDPGT

Localization, the ability of an unmanned robot to know its current position with respect to a reference, is very important for autonomous operations. The Global Positioning System (GPS) is a satellite-based navigation system that provides location information with respect to earth frame. GPS is one of the commonly used localization system for outdoor environments with an accuracy of +/-4m. However GPS signals are not available in all locations. This section is about a task that requires unmanned robots to reach an unknown location in GPS denied environments, similar to the conditions of search and rescue missions. Two robots, one aerial robot and one ground robot, are utilized to complete this task. The robots use only computer vision algorithms for localization and target identification. Both the robots use VDPGT for real-time path-generation and



(a) A-B (b) B-C

(c) C-D (d) D-A

Figure 4.13: Deviation from the actual path calculated for real-time experiments on AR Drone for Case – 2. The subplots show the deviation while the robot moves through different waypoints.

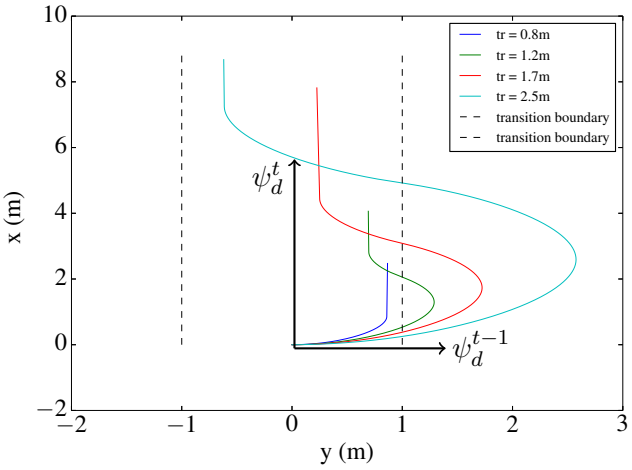


Figure 4.14: Path travelled by robots with non-zero minimum turning radius, while making a 90^0 turn, utilizing VDPGT. ψ_d^t and ψ_d^{t-1} denotes the desired orientation at the current and previous state.

path-following. The objective of this task is the demonstrate the effectiveness of VDPGT in realistic missions.

4.2.1 Vision-based Localization using Aerial and Ground Robots

Two robots, an AR Drone and Turtlebot, are used to complete the task. Details and specifications of Turtlebot and AR Drone are in Section 2.3.3 of Chapter 2. The robots initially start from a location in urban setting (an environment that has buildings), surrounded by buildings. The robots will not know their current location. The task for these robots is to navigate and successfully come out to a region without buildings. The aerial vehicle will fly on top providing an aerial view of the region below using its downward facing camera. The images from the camera are processed to identify the next possible target destination for the robots. The task is completed once the robots reach a region where there are no buildings nearby. Following are the assumptions made in order to narrow the task scope.

- The initial position of robots should always be in a region where there is a definite path for the ground robot to move (the robots should not be placed in isolated places that has no connections with any other region).
- The initial location of the robots should be such a way that the ground robot is always in the field of view of the bottom camera of the aerial robot.

4.2.1.1 Target Identification

(a) Top view (b) Front view

Figure 4.15: Top view and front view of the simulation environment

Fig. 4.15 shows the top view and front view of the operational environment utilized for simulation experiments. The steps for vision-based target identification

will be explained based on this environment. AR Drone flying on top provides a birds-eye-view of the environment below. AR Drone transmits the images from its bottom camera to the Turtlebot, where the image processing takes place. The color images provided by AR Drone are converted into greyscale images. The next step is to identify the edges in the images. However as the robots operate in an urban setting, there could be many edges. Hence image smoothing is carried out before proceeding to edge detection. The greyscale images are smoothed by applying a normalized box filter with 3×3 kernel. Canny Edge detection algorithm [126] is applied on the smoothened image to identify the edges in the image. Image thresholding is carried out to filter out unwanted edges (such as edges arising from the tiles of the roofs). After thresholding, the images are subjected to dilation and erosion [25] to identify the features in the processed image. Each pixel along the edges identified are iteratively expanded using a 2×2 kernel to connect the open edges. The dilated images are then eroded to normalize the images. The next step is to identify the target location for the robots to move, using the processed image from the aerial robot. Since the image obtained is from the bottom camera of the aerial vehicle, the optical centre of the image will always correspond to the position of the aerial vehicle (assuming that the bottom camera is fixed at the centre of the aerial vehicle). The images from the bottom camera of the aerial vehicle produces images with 640×480 px, the optical centre of the image is at 320×240px. Based on these two information, the target position for the robots in px will be at $t(u, v)$ where,

$$t_{forward}(u, v) = t(320\text{px}, v_{min}), \quad \forall \quad v_{min} \in [0, 240\text{px}), \tag{4.10}$$

$$t_{right}(u, v) = t(u_{max}, 240\text{px}), \quad \forall \quad u_{max} \in (320\text{px}, 640\text{px}], \tag{4.11}$$

$$t_{left}(u, v) = t(u_{min}, 240\text{px}), \quad \forall \quad u_{min} \in [0, 320\text{px}). \tag{4.12}$$

The maximum of the three $(t_{forward}, t_{right}, t_{left})$ is chosen as the target position. As the image matrix is not a square matrix, normalized weighting coefficients α_1, α_2 and α_3 are used for normalization. This normalization ensures that maximum

is not skewed to any side. The maximum is calculated by

$$t_{target} = max\left[(\alpha_1 \times t_{forward}), (\alpha_2 \times t_{right}), (\alpha_3 \times t_{left})\right] \tag{4.13}$$

In certain scenarios, the target found using Equation 4.13 may be infeasible due to noise from the image or the pathway to that target may be blocked. This is solved by expanding the search space $\chi_{(u,v)}$, using a $1 \times D_g$, where D_g is the diameter of the ground robot. The original search space is expanded from

$$\chi_{(u,v)} = \begin{cases} t_{forward}(u,v) \\ t_{right}(u,v) \\ t_{left}(u,v) \end{cases} \quad to \quad \chi'_{(u,v)} = \begin{cases} t'_{forward}(u,v) \\ t'_{right}(u,v) \\ t'_{left}(u,v) \end{cases} \tag{4.14}$$

where,

$$t'_{forward}(u,v) = t(D_g, v_{min}),\ \forall\ d \in \left[(320 - \frac{D_g}{2}), (320 + \frac{D_g}{2})\right], v_{min} \in [0, 240\text{px}), \tag{4.15}$$

$$t'_{right}(u,v) = t(u_{max}, D_g),\ \forall\ d \in \left[(240 - \frac{D_g}{2}), (240 + \frac{D_g}{2})\right],$$
$$u_{max} \in (320\text{px}, 640\text{px}] \text{ and} \tag{4.16}$$

$$t'_{left}(u,v) = t(u_{min}, D_g),\ \forall\ d \in \left[(240 - \frac{D_g}{2}), (240 + \frac{D_g}{2})\right], u_{min} \in [0, 320\text{px}). \tag{4.17}$$

The centre of $1 \times D_g$ kernel that has the maximum distance from the optical centre is chosen as the target position for the robots to move to. Fig. 4.17 shows the images obtained for all the processes mentioned above and Fig. 4.16h shows the final target position selected.

4.2.1.2 Target Localization

Once the target position in pixel coordinates is identified, the pin-hole camera model is utilized to convert the target position into real world coordinates. The

intrinsic matrix of a camera, using pin-hole camera model, is defined as

$$K = \begin{bmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.18}$$

where, $f_x,\ f_y$ are focal lengths of the camera and $x_0,\ y_0$ are principle offsets in x and y axis respectively. For a camera with a sensor of width W and height H that produces an image of width w and height h, the relationship between the pixel coordinates and real-world coordinates is given by

$$F_x = f_x \times \frac{W}{w}, \tag{4.19}$$

$$F_y = f_y \times \frac{H}{h}, \tag{4.20}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.21}$$

Hence using eqn. 4.21 the target location in pixel coordinates $t(u, v)$ is converted using real-world coordinates $t(x, y)$ through

$$t_x = (u - x_0) \times \frac{z}{f_x}\ and \tag{4.22}$$

$$t_y = (v - y_0) \times \frac{z}{f_y}. \tag{4.23}$$

As the aerial vehicle is always flown at a constant height from the ground z is always known. Once the target position for the ground robot in real-world coordinates is computed using eqns. 4.22 and 4.23, these values are utilized by VDPGT algorithm to generate path.

It is common that the aerial vehicle may fly faster than the ground robot leading to large difference in positions. In these scenarios, it is possible that the

target position identified may be obstructed for the ground robot if the shortest path is followed (as shown in Fig. 4.17a). This problem is overcome by identifying intermediate waypoints before proceeding to the target (as shown in Fig. 4.17b).

4.2.1.3 Modes of Operation

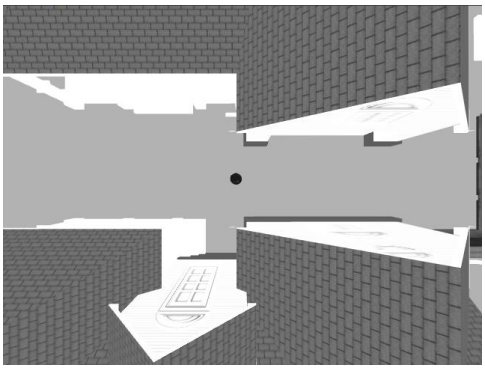
The robots can operate in two modes: autonomous and semi-autonomous modes. In autonomous mode, both aerial and ground robots are fully autonomous. Both the robots use VDPGT to move to the target location and explore the unknown environment. On the other hand, in semi-autonomous mode, the aerial robot is controlled by a ground operator and the ground robot moves autonomously. Semi-autonomous mode is useful in cases where a particular area is to be explored. The aerial robot is moved by the operator while the ground robot follows the target positions created. Table. 4.1 shows the exploration strategies followed by the ground robot for both autonomous and semi-autonomous modes of operations.

Table 4.1: Exploration strategy followed by the ground robot for coordinated vision-localization

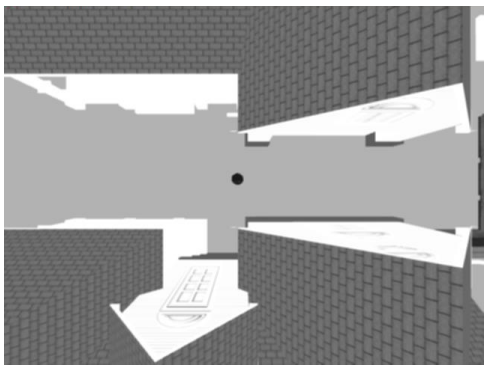
Situation	Action
No obstacles in any direction (open space)	Move forward
Obstruction only in front of the robot	Move left
Obstruction only in left of the robot	Move forward
Obstruction only in right of the robot	Move forward
Obstruction in left and right of the robot	Move forward
Obstruction in front and left of the robot	Move right
Obstruction in front and right of the robot	Move left
Obstructions in front, left and right of the robot	Turn 180^0 back and move forward

4.2.1.4 Simulation and Experimental Results and Discussion

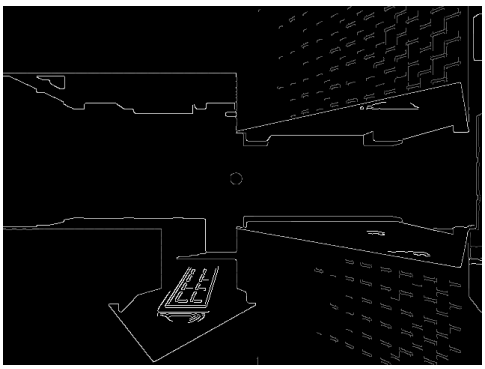
In Fig. 4.18, figs. 4.18a and 4.18b show the environments utilized for simulation and real-time experiments respectively. Turtlebot and AR Drone are utilized for both simulation and real-time experiments. The task for the robots in simulation experiments is to navigate successfully out of the region with houses. The robots should make two 90^0 turns to achieve the target. Fig. 4.19 shows the results for semi-autonomous mode. Fig. 4.19a and 4.19b show the path travelled by



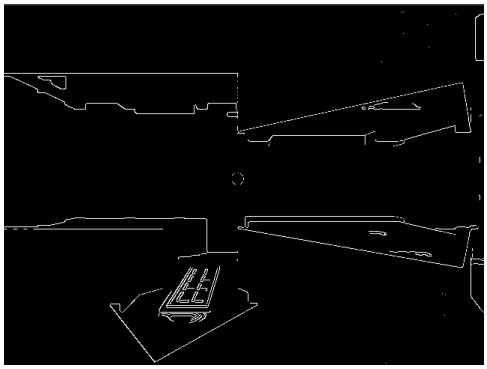
(a) Greyscale image



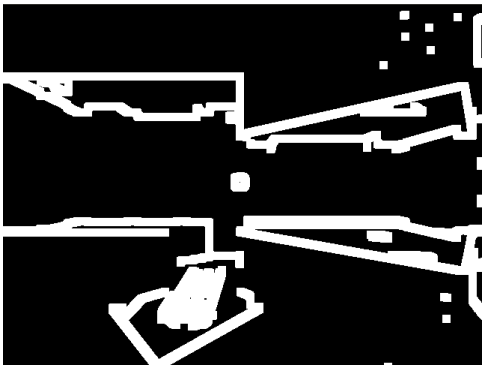
(b) Smoothened image



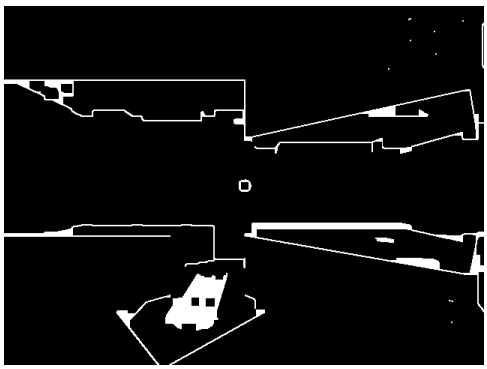
(c) Canny image



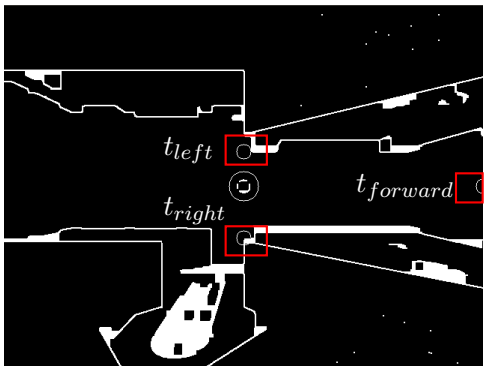
(d) Thresholded image



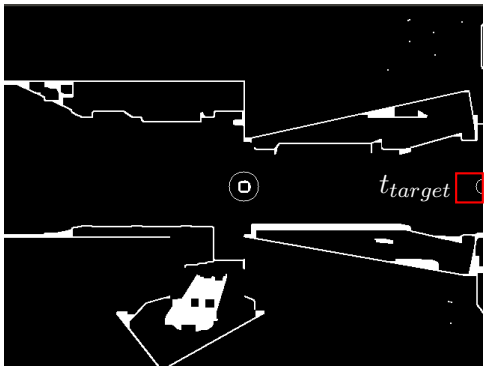
(e) Dilated image



(f) Eroded image

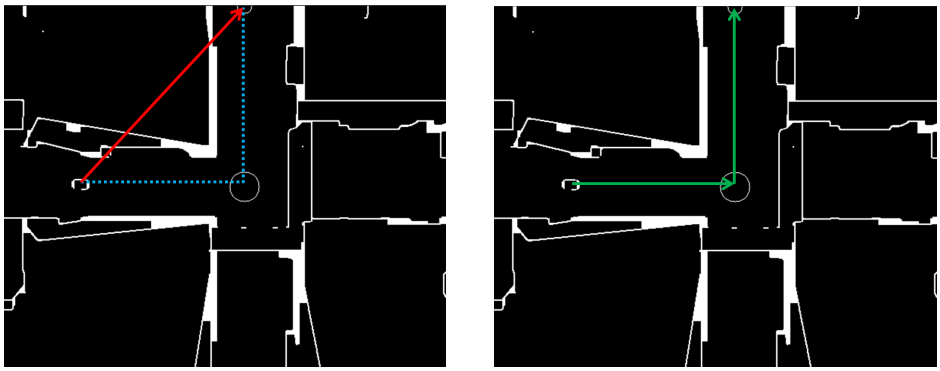


(g) Potential targets



(h) Final target identified

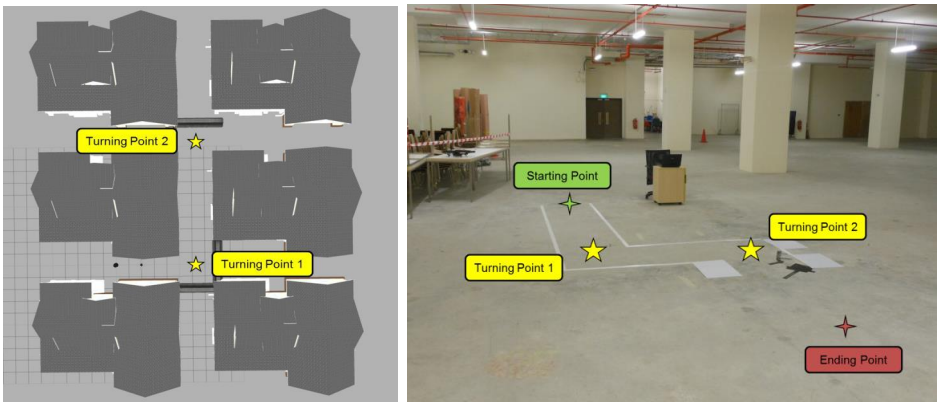
Figure 4.16: Sequence of steps involved in computer vision based target identification.



(a) Infeasible path generated when the aerial robot is far away from the ground robot

(b) Feasible path generated by adopting intermediate waypoints

Figure 4.17: Images showing the scenario where an infeasible path is generated when the aerial vehicle is far away from the ground robot. The new feasible path is generated by adopting an intermediate waypoint.



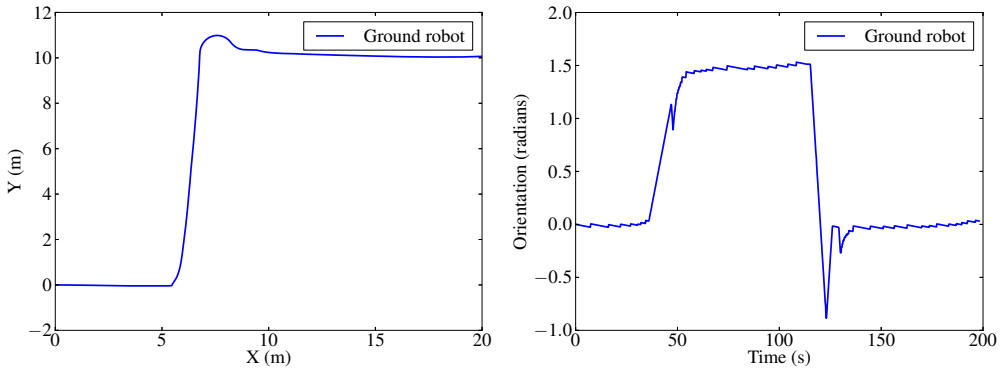
(a) Simulation environment

(b) Environment for real-time experiments

Figure 4.18: Simulation and real-time environments utilized for the experiments

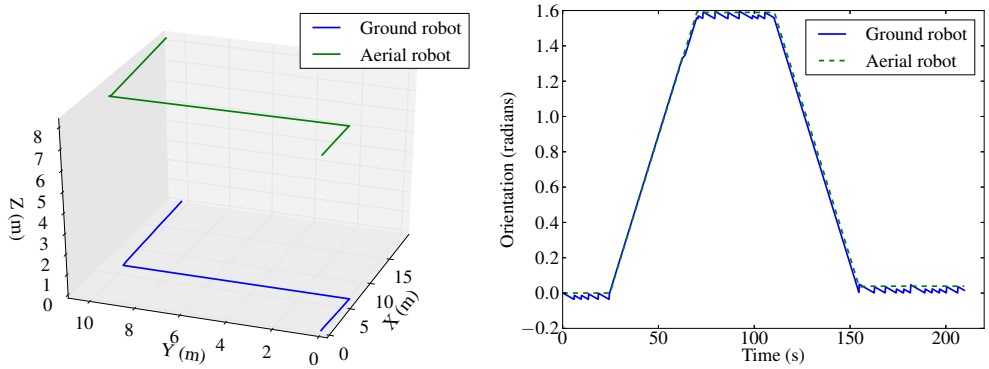
the ground robot and change in orientation of the ground robots, operating in semi-autonomous mode. It can be seen clearly that the ground robot is able to follow the aerial robot and achieve the target successfully. Fig. 4.20 shows the results when the robots are operating in autonomous mode. It can be seen from Fig. 4.20a that the robots using VDPGT are capable of completing the task autonomously. The real-time environment is created to replicate the operating conditions similar to the simulation environment (conditions that require the robots to make two 90^0 turns). The robots operate in semi-autonomous mode due to hardware limitation of the aerial robot. Fig. 4.21 shows the path travelled

by the ground robot while successfully completing the mission in real-time. Both the simulation and real time experiments demonstrate the usefulness of VDPGT.



(a) Path travelled by the ground robot (b) Change in orientation (Yaw angle)

Figure 4.19: Simulation results for semi-autonomous mode

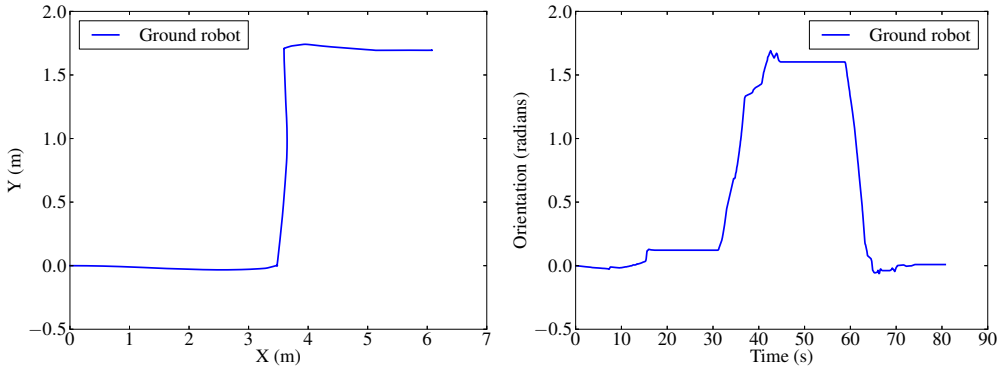


(a) Path travelled by the ground robot and aerial robot (b) Change in orientation (Yaw angle)

Figure 4.20: Simulation results for autonomous mode

4.3 Incorporating Path-Generation and Path-Following Behaviours into SOIFRA

Overall architecture of SOIFRA, a service oriented behaviour-based multi-agent framework for unmanned aerial and ground vehicles, is explained in Section. 2.2 of Chapter 2. SOIFRA framework is incorporated with collision avoidance behaviour, a basic behaviour required for autonomous operation of mobile robots,



(a) Path travelled by the ground robot (b) Change in orientation (Yaw angle)

Figure 4.21: Real-time results for semi-autonomous mode

as explained in Section. 3.2 of Chapter 3. This section deals with incorporating VDPGT algorithm into SOIFRA to provide path-generation and path-following behaviours. As explained in Section. 2.2 SOIFRA is made up of deliberation, behaviour and execution layers. Goal generator, planner-matcher and agents are part of deliberation layer while agent services and actions carried out by the agents are part of behaviour and execution layer respectively. The overall goal for this mission is to reach a specific location in an unknown environment with obstacles. The overall goal is divided into two sub-goals. A sub-goal to achieve collision avoidance and a sub-goal to take the robot to the target location. Architectural overview of SOIFRA for the given mission is shown in Fig. 4.22.

The mission under consideration is similar to the mission discussed in Section. 3.2. The requirement that the robots are to reach a specific location is the difference between the two missions. Hence the framework for this mission can be considered as an advancement to the framework discussed in Section. 3.2. The steering agent (Str:AGT) navigating the robot and obstacle agent (Obs:AGT) performing obstacle detection and avoidance are carried over for this mission. The services offered and actions performed by Str:AGT and Obs:AGT are explained in Section. 3.2. In addition to the Str:AGT and Obs:AGT there are two more agents that are developed to achieve this mission. An agent to generate path and an agent to follow the generated path are the agents developed. Let PF:AGT and PG:AGT denote the path-follow agent and the path-generation

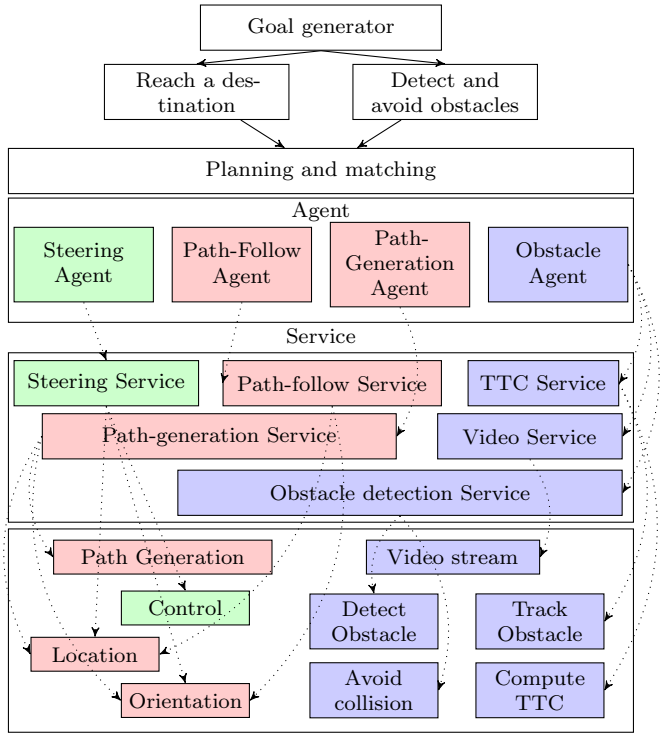


Figure 4.22: Architectural overview of SOIFRA with path-generation, path-following and collision avoidance.

agent respectively. PG:AGT provides path-generation service (pg:SRV) while PF:AGT provides path-follow service (pf:SRV). PG:AGT generates the shortest path to reach the given target in real-time utilizing the actions to generate path (gpath:ACT) and measure the position (pos:ACT) and orientation (ori:ACT) of the robot. Once a path is generated, PG:AGT publishes the desired orientation ψ_d for the robot on *rostopic*. This desired orientation is subscribed by PF:AGT. The path-follow service (pf:SRV) of PF:AGT generates the control signals that are required to direct the robot towards the path generated by PG:AGT, utilizing VDPGT. If the robot crosses a transition boundary while moving towards the target destination, PG:AGT generates the new path to reach the target. The action sequences of the path-generation and path-following operations are shown in Fig. 4.23 and Fig. 4.24

Obstacle agent (Obs:AGT), responsible for obstacle detection and avoidance, initiates the actions of video:SRV, TTC service (TTC:SRV) and obstacle detection service (det:SRV). Obs:AGT utilizes TTC-EO, explained is Section. 3.1.2.3, for

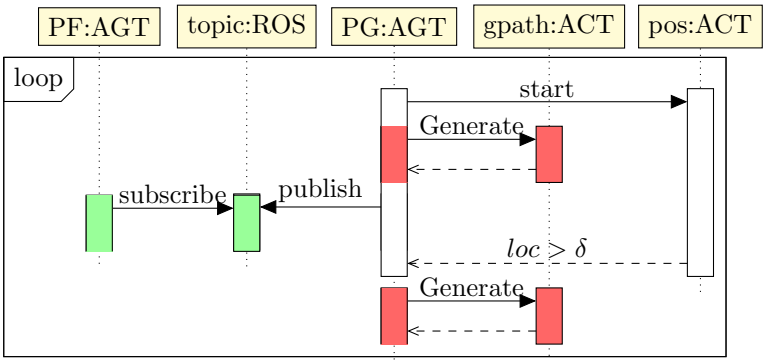


Figure 4.23: Sequences for path-generation. AGT and ACT represent agent and actions. The path-generation agent PG:AGT, generates a path for the robot to follow using path-generation action gpath:ACT and utilizes pos:ACT to monitor the current position of the robot. The generated path is communicated to the path-follow agent PF:AGT thorough, *rostopic*. If the robot crosses a transition boundary a new path is generated.

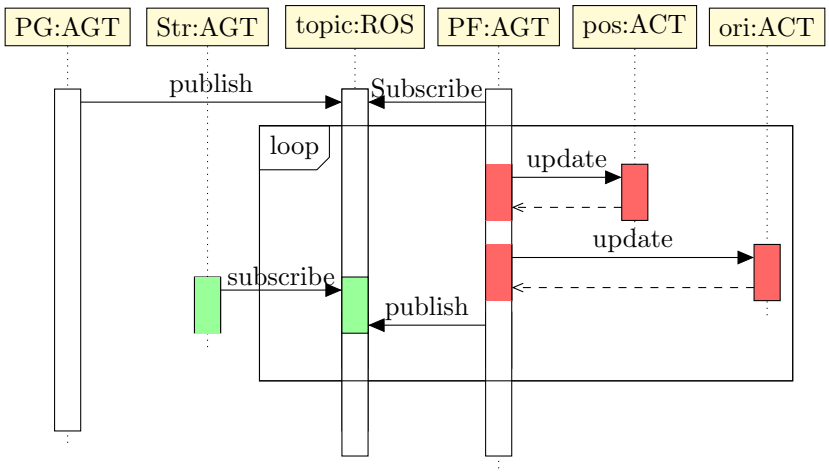


Figure 4.24: Sequences for Path-following. AGT and ACT represent agent and action respectively. The path-follow agent PF:AGT receives the path to follow from path-generation agent PG:AGT thorough *rostopic*. Position and orientation of the robot obtained through pos:ACT and ori:ACT actions are utilized in generating control actions for the robot. The generated control parameters are intimated to the steering agent Str:AGT.

collision avoidance. Obstacle agent (Obs:AGT) and path-follow agent (PF:AGT), publish control velocities for the robot on *rostopic*. Str:AGT, once initialized, assigns priority index for obstacle agent and path-follow agent. Str:AGT subscribes to the topics published by both Obs:AGT and PF:AGT, but only the action with a higher priority is executed. Str:AGT utilizes the control velocities obtained from PF:AGT as PF:AGT has higher priority when no obstacle is detected on the robot's path. The priority index for Obs:AGT is zero when distance to

the obstacle detected is greater than the Critical distance λ and is the highest when distance to the obstacle is less than λ. Critical distance, the minimum distance to the obstacle within which action must be taken to avoid an obstacle, is determined based on the size and linear velocity of the robot, is fixed at 2.5m for the obstacle avoidance experiments conducted. The control velocities computed are published to control the velocity of the robot. Once ROS parameter server is updated by Obs:AGT, (when distance to the obstacle is less than λ), Str:AGT publishes the control velocities obtained from Obs:AGT and initiates the action to monitor the position of the robot. Once the obstacle is avoided, the Str:AGT informs to Obs:AGT that obstacle is avoided. The obstacle agent then updates the PS:ROS which result in PF:AGT regaining higher priority. The sequences for collaborative actions performed by the steering agent (Str:AGT) is shown in Fig. 4.25.

4.4 Simulation and Real-time Experiments Demonstrating Path-Generation and Path-following Behaviour of SOIFRA

This section presents the experimental and simulation results for the mission undertaken. Robot's mission is to reach a known destination in an unknown environment. The robot exhibits obstacle avoidance and path-following behaviours to complete the mission. Two different robotic platforms, Turlebot and an AR Drone are utilized to demonstrate the interoperability of SOIFRA. The operational environments for simulation and real-time experiments are same as the environment utilized for experiments in Section. 3.3. Fig. 3.11 and Fig. 3.15 show the operational environments for the simulation and real-time experiments while Fig. 4.26 shows the layout (top view) of the operational environment.

The simulation and experimental results for Turtlebot and AR drone are explained in the following subsections. The two robots complete the same mission of reaching a destination in an unknown environment, individually. The critical

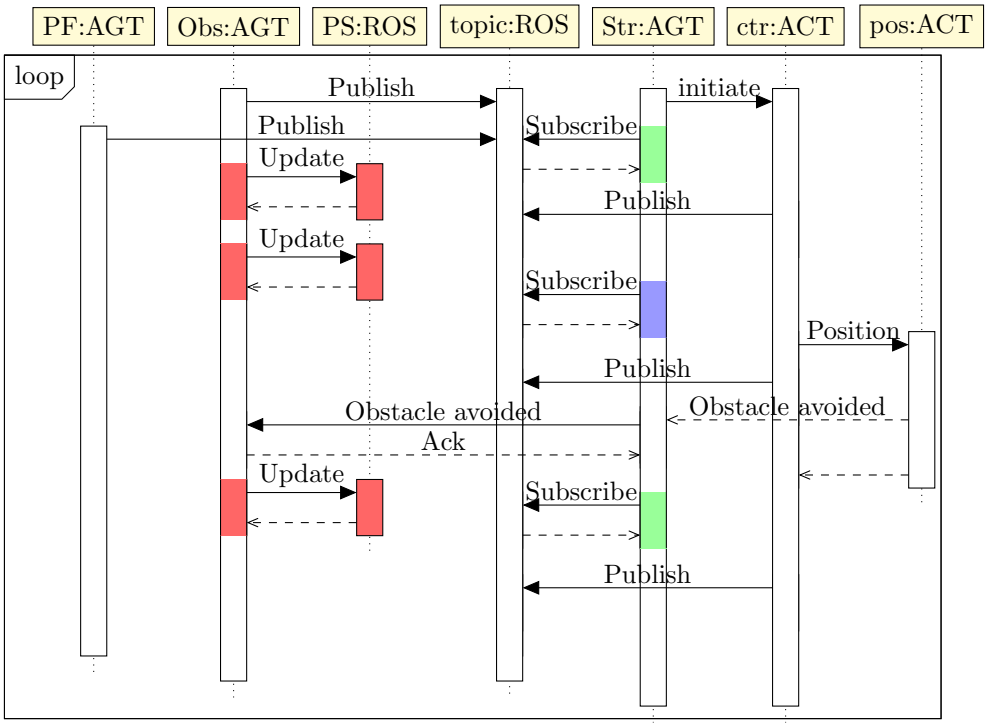


Figure 4.25: Operational sequences demonstrating collaboration between path-follow agent PF:AGT and obstacle agent Obs:AGT. AGT and ACT represent agent and action respectively. The Steering agent Str:AGT receives inputs from PF:AGT and Obs:AGT simultaneously through *rostopic*. When there is no obstacle detected along the robot path, PF:AGT has higher priority. Once distance to an obstacle detected is less than the critical distance λ, Obs:AGT is assigned the highest priority. Once obstacle is avoided, PF:AGT regains higher priority.

distance λ is fixed at 2.5m and the transitional boundary δ is fixed at +/-1m from the starting point (along Y axis). Robot turns left if the obstacle is located on the right side of the optical axis of the robot's camera and turn right otherwise. The robots have zero turning radius in simulation experiments, while for real-time experiments the robots are forced to turn with non-zero turning radius.

4.4.1 Simulation and Real-time experimental Results for Turtlebot

Velocity of Turtlebot is fixed at 0.1m/s in simulation and in real-time experiments. The Turtlebot encounters an obstacle at 5m from the starting location in simulation and 4.5m from the starting location in the real-time experiment. The simulation and experimental results obtained for the case study are shown

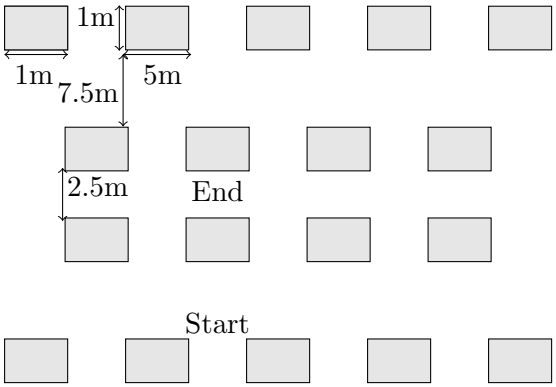
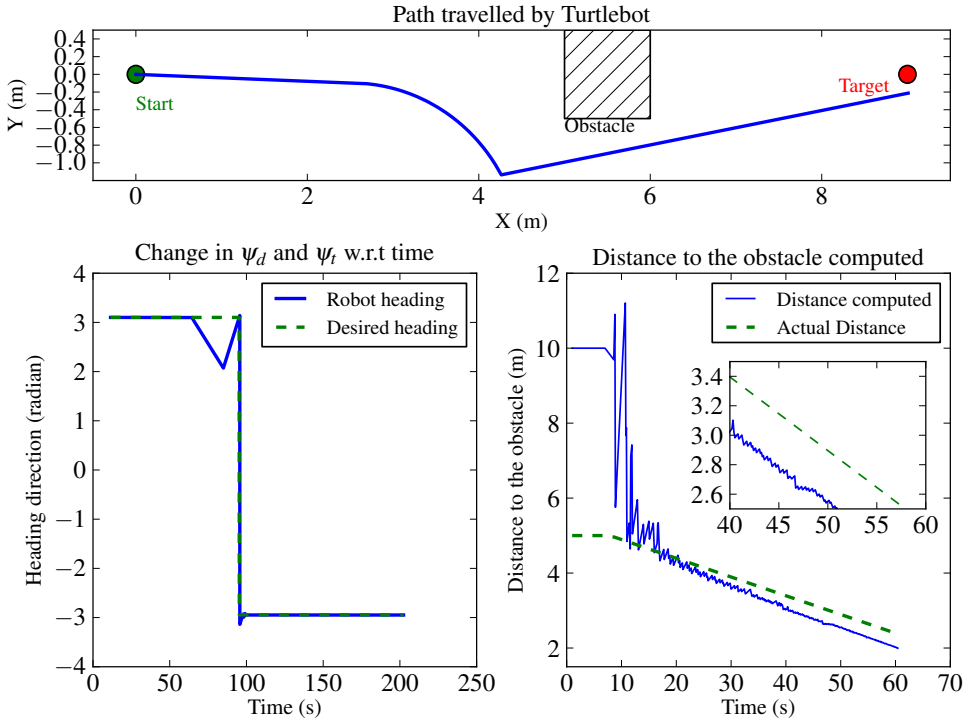
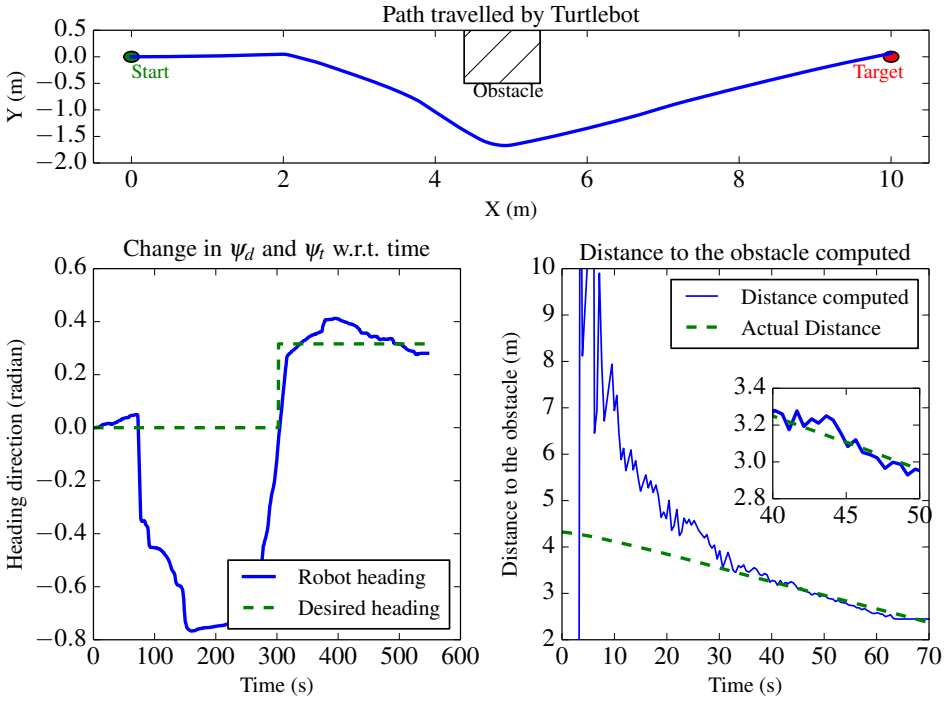


Figure 4.26: Layout of the operational environment (not drawn to scale). The mission is completed once the robot reaches the destination (end).

in Fig. 4.27. In Figs. 4.27a and 4.27b, the start and target denote the starting location and the destination for the robot. The distance to the obstacle, estimated based on expansion of object method (TTC-EO), is utilized for obstacle avoidance. The robot moves towards its destination initially when there is no obstacle (distance to the obstacle plots in Fig. 4.27 (time period $0 - 70s$ for simulation and real-time experiments). Once the distance to the obstacle is less than λ, obstacle avoidance algorithm is initiated (change in orientation plots in Fig. 4.27) (time period $70 - 90s$ for simulation and $70 - 350s$ for real-time experiments). At the critical distance, the obstacle to be avoided is on the left side of the optical axis of Turtlebot's camera. As a result, Turtlebot turns right in both simulation and real-time experiments. As the robot crosses a transition boundary at the end of obstacle avoidance process, the path-planning agent generates a new path. This is evident by the change in desired orientation of the robot (desired orientation changes from 3 to -3 radians in simulation while it changes from 0 to 0.3 in real-time experiments). As the simulation experiment assumes that Turtlebot has zero turning radius, the robot first changes its orientation to match the new desired orientation and undergoes translation, while in real-time experiment, the robot undergoes both change in orientation and translation at the same time because of its non-zero turning radius. As explained in Algorithm 4.1, the robot stops when it reaches the stopping criteria, $p^* > 1$ is reached. Figs. 4.27a and 4.27b show that the robot is able to follow a shorter path, avoid obstacle and



(a) Simulation results



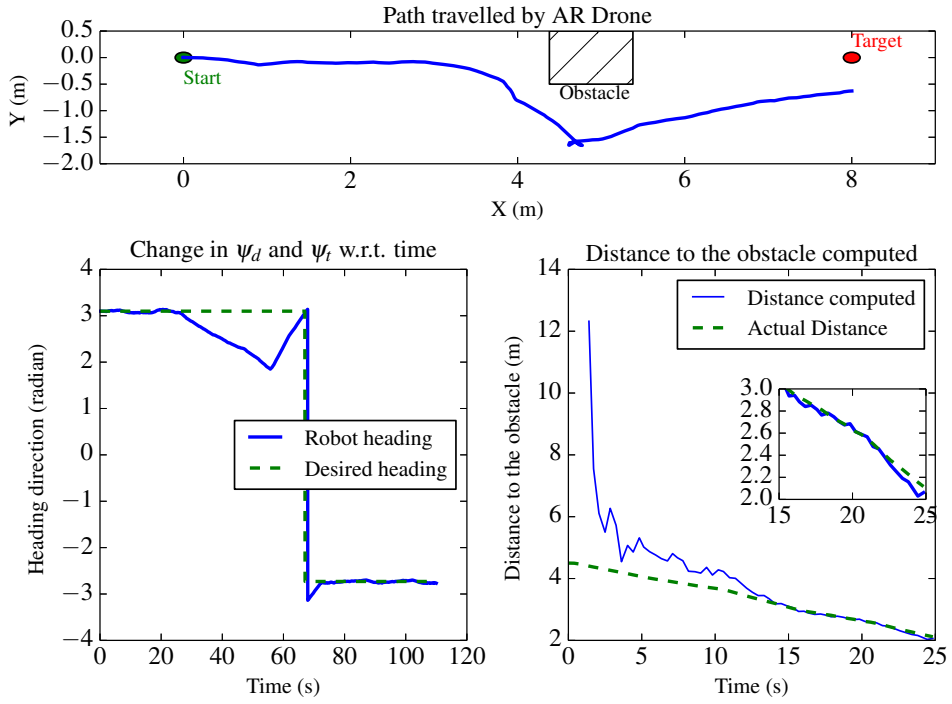
(b) Results from real-time experiments

Figure 4.27: Simulation and experimental results for the case study undertaken using a Turtlebot. Turtlebot moves towards the target in an unknown environment. Distance to the obstacle is estimated utilizing the expansion of object based time-to-contact method (TTC-EO).

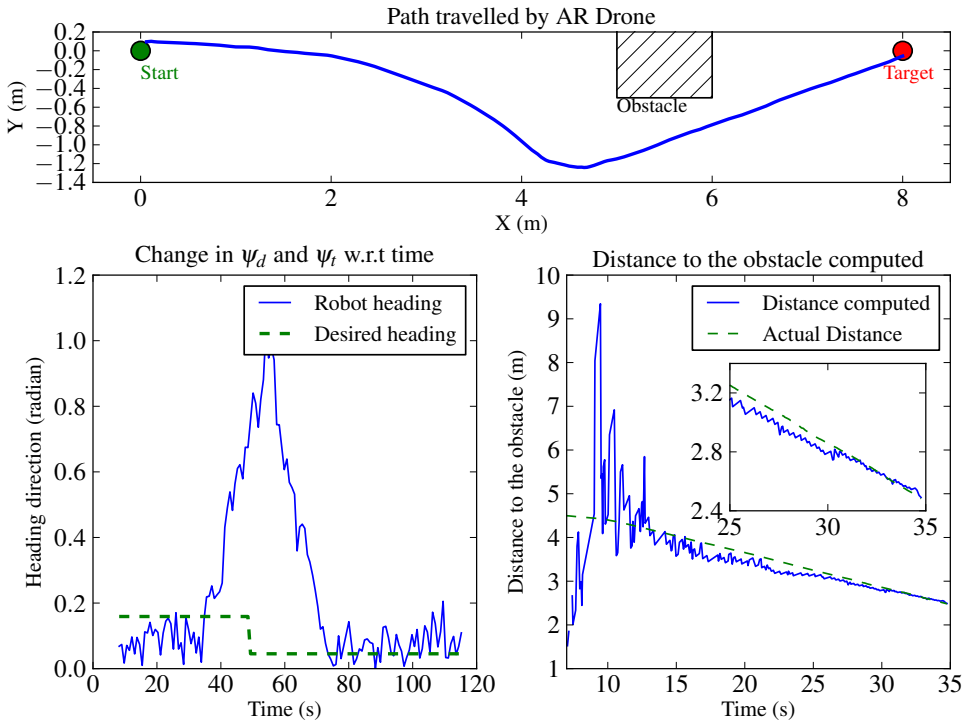
reach the desired destination successfully in simulation and real-time experiments.

4.4.2 Simulation and Real-time Experimental Results for AR Drone

AR Drone encounters an obstacle around 4.5m and 5m from its initial location in simulation and real-time experiments respectively. AR Drone travels at 0.1m/s in simulation and 0.65m/s in real-time experiments (observed lowest velocity for following straight line trajectories). Fig. 4.28 shows the simulation and experimental results for the case study. In Figs. 4.28a and 4.28b start and target denote the starting location and destination for AR Drone. Obstacle avoidance algorithm utilizes the distance estimated from the expansion of object based time-to-contact method (TTC-EO). The robot moves towards its destination initially when there is no obstacle to be avoided (distance to the obstacle plots in Fig. 4.28) (time period $0 - 20s$ for simulation and $0 - 35s$ for real-time experiments). Once the distance to the obstacle is less than λ, obstacle avoidance algorithm is initiated (change in orientation plots in Fig. 4.28) (time period $20 - 70s$ for simulation and $35 - 60s$ for real-time experiments). At the critical distance, the obstacle to be avoided is on the left side of the optical axis of AR Drone's camera. As a result, AR Drone turns right in both simulation and real-time experiments. At the end of obstacle avoidance process, in simulation and real-time experiments, the change in desired orientation of the robot confirms that AR Drone crosses a transition boundary (desired orientation changes from 3.14 to -3 radians in simulation while it changes from 0.19 to 0.05 in real-time experiments). This results in path-planning agent generating a new path at the end of obstacle avoidance process. As the simulation experiment assumes that AR Drone has zero turning radius, the robot first changes its orientation to match the new desired orientation and undergoes translation, while in real-time experiment, the robot undergoes both change in orientation and translation at the same time because of its non-zero turning radius. As explained in Algorithm 4.1, the robot



(a) Simulation results



(b) Results from real-time experiments

Figure 4.28: Simulation and experimental results for the case study undertaken using an AR Drone. AR Drone moves towards the target in an unknown environment. Distance to the obstacle is estimated utilizing the expansion of object based time-to-contact method (TTC-EO).

stops when the stopping criteria, $p^* > 1$ is reached. Figs. 4.28a and 4.28b show that the robot is able to follow a shorter path, avoid obstacle and reach the desired destination successfully in simulation and real-time experiments.

4.5 Summary

This chapter presents an platform independent, real-time path-generation and path-following algorithm for autonomous ground and aerial vehicles. The central focus of this chapter is to incorporate VDPGT algorithm into SOIFRA framework to provide path-generation and path-following behaviours. The performance of VDPGT algorithm developed is demonstrated through two separate simulation and real-time experiments with UAV and UGV. The usefulness of VDPGT is demonstrated by utilizing VDPGT to complete a realistic task in both simulation and real-time experiments. Both UAV and UGV are able to localize utilizing only computer vision techniques and successfully navigate to their destination utilizing VDPGT.

Two agents, an agent to provide path-generation and an agent to provide path-following functionality are developed and incorporated into SOIFRA. To demonstrate the path-following ability of SOIFRA a mission where a robot is to reach a specific destination, in an unknown environment, is conducted. The real-time path-planning and tracking algorithm, VDPGT, and TTC-EO collision avoidance algorithm are utilized on both aerial and ground robots for completing the mission. Performance of VDPGT is demonstrated through multiple path tracking experiments on a Turtlebot and AR Drone. The results show that the behaviour based nature of SOIFRA allows the agents to dynamically update their knowledge in real-time leading to effective collision avoidance and successful path tracking. SOIFRA achieves interoperability across diverse robotic platforms such as aerial and ground robots even though the control mechanisms for the robots are different. This proves that SOIFRA is able to successfully incorporate platform independent algorithms.

Chapter 5

Real-time Pedestrian Tracking, with SOIFRA, Utilizing Human Visual Cortex Model

"Effort only fully releases its reward after a person refuses to quit "

– Napolean Hill

Unmanned robots as explained in previous chapters are robots capable of intelligent actions and motions without any guidance from a human or teleoperator. IRobot's Packbot [146], Predator aircraft, Mars exploration rovers: Spirit and Opportunity [86] and Aerosonde [49] are a few examples of autonomous robots. In order to account for safety, autonomous robots were operated in human isolated environments. Due to recent advancements in robotics research, autonomous robots are increasingly utilized for various applications in environments where human presence is unavoidable. Autonomous lawn mover [95], IRobot's Roomba and Scooba [61], autonomous vacuum cleaner and floor washer robots are a few robots that attempt to relieve home-owners from some of their everyday housework. Kiva warehouse-management system [145], Grey-Orange's Butler robotic system, Hope Technik's Sesto are a some of the recently developed Autonomous Guided Vehicles (AGV) utilized for warehouse-management. Self-

driving or driver-less cars, researched upon by Google, General Motors, Tesla Motors and Ford Motors, are autonomous cars that are designed to coexist and cooperate with humans. Autonomous industrial robots generally perform tasks rapidly with many fast-moving parts creating an unsafe environment for humans to coexist. Recently there are a few works that aim to improve the cooperation between humans and autonomous robots in industrial environments [21, 37, 139]. Autonomous robots, mobile as well as industrial manipulators, include perception systems to actively detect humans while operating in a human coexisting environment. Human detection and human collision prediction are important factors to ensure safe human-robot coexisting conditions.

Pedestrian detection is the ability of a system to detect humans in the environment through active perception. Most of the existing pedestrian detection systems utilize vision sensors in various configurations. The vision sensors could be based on visible light as well as infrared radiation. Pedestrian detection solutions based on time-of-flight sensors such as radar and laser range scanners are also available. This chapter explains about a human visual cortex-based pedestrian detection algorithm developed, that improves the performance of existing state-of-the-art pedestrian detectors. The developed human visual cortex-based pedestrian detector is incorporated into SOIFRA to provide pedestrian detection ability for UAVs and UGVs.

5.1 Mathematical Model of a Human Retina

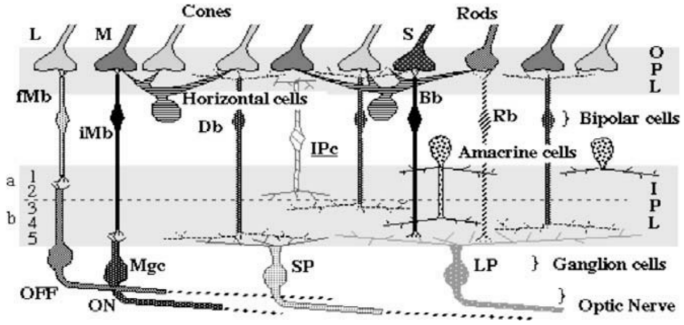
The human visual cortex/ retina model is a biologically inspired model of human retina. It is useful in developing efficient and fast computer vision algorithms for low level image processing. This retina model is modified and utilized for pedestrian detection in this work. Biological architecture of a human retina and an overview about modelling different components of the retina are provided in this section. Detailed explanations of the retina model utilized are discussed in [9, 11, 48].

5.1.1 Human Retina

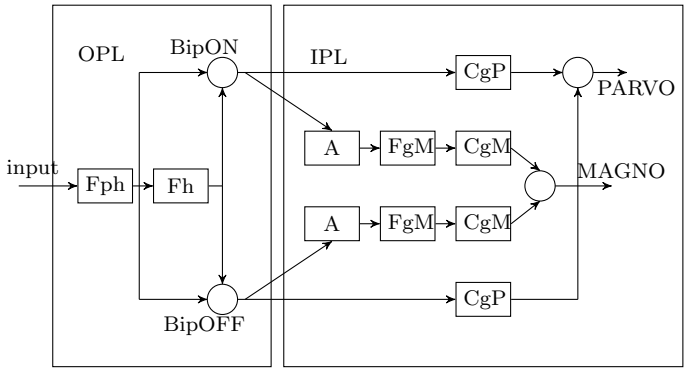
A human retina is composed of different retina cells such as photoreceptors, horizontal cells, bipolar cells, ganglion cells and amacrine cells. In Fig. 5.1, Figs. 5.1a and 5.1b shows the biological architecture and model of a human retina respectively. Photoreceptors are responsible for visual data acquisition and local logarithmic compression of the image luminance. Horizontal cells integrate and regulate input from multiple photoreceptor cells. Signals from photoreceptors or horizontal cells are transmitted to ganglion cells through bipolar cells. Ganglion cells transmit image–forming and non–image forming visual information from the retina. Amacrine cells are responsible for creating functional subunits within the ganglion cell layer enabling ganglion cells to observe a small dot moving a small distance. Cells of retina, which are connected to each other, are separated into two layers: the Outer Plexiform Layer (OPL) and the Inner Plexiform Layer (IPL). Information from multiple channels of IPL layer constitutes the output of the retina, of which Parvocellular channel (Parvo) and Magnocellular channel (Magno) are important. Parvocellular channel, most present at the fovea level (central vision) of the retina, is dedicated to detail extraction while Magnocellular channel, most present outside of the fovea (peripheral vision) of the retina, is dedicated to motion information extraction. As detail and motion data of the same region are available in parallel, information from Parvo and Magno channels is useful for computer vision algorithms.

5.1.2 Photoreceptors Model

Photoreceptors are involved in visual data acquisition and local logarithmic compression of the image luminance. Photoreceptors have the ability to adjust their sensitivity with respect to the luminance of their neighborhoods [9, 11, 48]. Michaelis-Menten relation [9] normalized for a luminance range of [0, V_{max}] is



(a) Biological architecture [48]



(b) Model of retina

Figure 5.1: Biological architecture of a human retina

utilized to model the ability of photoreceptors.

$$C(pr) = \left[\frac{R(pr)}{R(pr)+R_0(pr)}\right] V_{max} + R_0(pr), \tag{5.1}$$

$$R_0(pr) = V_0 L(pr) + V_{max}(1-V_0), \tag{5.2}$$

where, $C(pr)$ is the adjusted luminance of the photoreceptor pr. $C(pr)$ depends on the current luminance $R(pr)$ and the compression parameter $R_0(pr)$ linked linearly to the local luminance $L(pr)$ of the neighborhood of the photoreceptor pr. V_0 is the contribution of a static compression parameter of the range [0,1]. Local luminance ($L(pr)$) is computed by applying a spatial low pass filter to the input image, achieved through horizontal cells.

5.1.3 Outer Plexiform Layer Model

OPL is modelled as non-separable spatio-temporal filter with a spatial frequency f_s and temporal frequency f_t [9, 11, 48]. The spatio-temporal filter, modelling the

cellular interactions of the OPL, is considered as a difference between the spatio-temporal filters that model the photoreceptor network ph and the horizontal cell network h. The transfer function of OPL filter is given as

$$F_{OPL}(f_s, f_t) = F_{ph}(f_s, f_t).[1 - F_h(f_s, f_t)], \tag{5.3}$$

$$F_{ph}(f_s, f_t) = \frac{1}{1 + \beta_{ph} + 2\alpha_{ph}.(1 - cos(2\pi f_s)) + j2\pi\tau_{ph}f_t}, \tag{5.4}$$

$$F_h(f_s, f_t) = \frac{1}{1 + \beta_h + 2\alpha_h.(1 - cos(2\pi f_s)) + j2\pi\tau_h f_t}, \tag{5.5}$$

where, β_{ph} and β_h are gains of the spatio temporal filters F_{ph} and F_h respectively. α_{ph} and α_h are the high and low cut-off frequencies, while τ_{ph} and τ_h are the temporal filtering constants. The difference between F_{ph} and F_h, the positive and negative parts of the difference between P_h and h images are represented by $BipON$ and $BipOFF$ operators. This models the actions of bipolar cells which divide the OPL outputs into two channels.

5.1.4 Model of Parvo and Magno Channel in Inner Plexiform Layer

Parvocellular channel dedicated to detail extraction is mostly present in the central visual region of the retina in IPL. Magnocellular channel mostly present at peripheral vision of the retina in IPL, is dedicated to motion information extraction. This subsection explains the models of both Parvocellular and Magnocellular channels.

5.1.4.1 Paravocellular Channel

Contour information from $BipON$ and $BipOFF$ operators of OPL reach the ganglion cells of the Parvo channel. These information is locally enhanced which reinforces the contour data. This ability of the ganglion cells in Parvo channel CgP is modelled using Michaelis-Menten law similar to the photoreceptors [122].

5.1.4.2 Magnocellular Channel

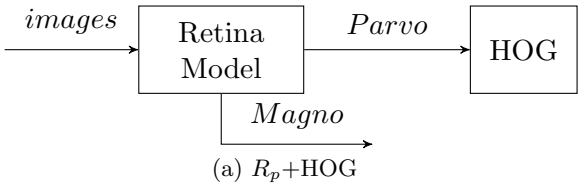
Magnocellular channel of IPL contains the amacrine cells that act as high pass temporal filters. High pass temporal filtering effect is modelled utilizing a first order filter [11].

$$A(Z) = b_c \left[\frac{1 - z^{-1}}{1 - bZ^{-1}} \right] \qquad with \quad b_c = e^{\frac{-\Delta t}{\tau_A}} \tag{5.6}$$

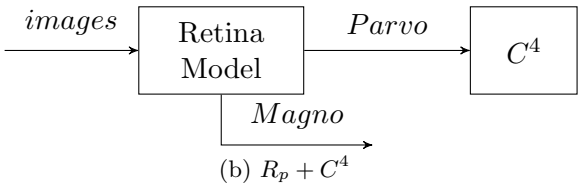
where, Δt is the discrete time step and τ_A is the time constant of the filter. The amacrine cells (A) are connected to the bipolar cells and as on the Paravo channel, perform local contrast compression (CgM) as well as act as low pass spatial filter (FgM similar to the filters of the OPL model).

5.2 Pedestrian Detection Utilizing Retina Model

Pedestrian detection utilizing a retina model is performed by combining information from both the Parvo and Magno channel of the retina. This section explains how information from Parvo and Magno channel are processed and combined to perform pedestrian detection. Retina filter is applied to the incoming images and output from Parvo and Magno channel are processed separately.



(a) R_p+HOG



(b) $R_p + C^4$

Figure 5.2: Overview of $R_p + HOG$ and $R_p + C^4$ pedestrian detectors

5.2.1 Processing Information from Parvo Channel

The Parvo channel of retina is useful in extracting detailed information from an image. The output from Parvo channel of retina model usually contains images with locally enhanced contours. These information are combined with an existing pedestrian detection algorithm to detect pedestrians. HOG and C^4 based pedestrian detectors are the best performing feature-based pedestrian detectors. As HOG and C^4 are two standard pedestrian detection algorithms, the processed image from Parvo channel are combined with HOG ($R_p + HOG$) or C^4 ($R_p + C^4$) pedestrian detection algorithms (Fig. 5.2). Output from $R_p + HOG$ and $R_p + C^4$ are compared with outputs from standard HOG and C^4 algorithms. INRIA, MIT and Caltech datasets are utilized for this comparison study. Following are the performance measures utilized for the comparison study:

- Number of False Negatives (FN): The total number of pedestrians present in the image, not detected,
- Number of False Positives (FP): Total number of pedestrians not present in the image, detected,
- Number of True Positives (TP): Total number of pedestrians present in the image, detected,
- False Positive Rate (FPR): The number of false positives (FP) divided by the sum of number of true positives (TP) and false positives (FP),
- False Negative Rate (FNR): The number of false negatives (FN) divided by the sum of number of true positives (TP) and false negatives (FN),
- Detection Rate (DR): The number of true positives (TP) divided by the sum of true positives (TP) and false negatives (FN) and
- Detection Precision (DP): The number of true positives (TP) divided by the sum of the true positives (TP) and false positives (FP).

FP, FPR and DP are not applicable for MIT database as negative samples are not available in MIT database. Tables 5.1, 5.2 and 5.3 show the performance

Table 5.1: Performance measures from MIT Dataset

	HOG	$R_p + HOG$	C^4	$R_p + C^4$
FN	513	143	386	279
TP	411	781	538	645
FNR	0.55519	0.15476	0.41775	0.30195
DR	0.44481	0.84524	0.58225	0.69805

of HOG, $R_p + HOG$, C^4 and $R_p + C^4$ pedestrian detectors on MIT, INRIA and Caltech databases respectively. Table 5.1 highlights the significance of retina filter for pedestrian detection. Standard HOG and C^4 pedestrian detectors have a detection rate of 0.44 and 0.58 respectively while the same algorithms when combined with retina filter ($R_p + HOG$ and $R_p + C^4$) have an improved detection rate of 0.84 and 0.69 respectively on MIT dataset. In INRIA and Caltech datasets, generally $R_p + HOG$ and $R_p + C^4$ outperforms standard HOG and C^4 significantly in most cases while their performances are comparable with standard HOG and C^4 algorithms in other instances. In Fig. 5.3, sub-figs. 5.3a and 5.3c shows the images from MIT dataset that HOG algorithm failed to detect while Figs. 5.3b and 5.3d show the successful pedestrian detection from $R_p + HOG$ for the same images. This highlights the importance of local contour enhancement provided by the Parvo channel of the retina filter in pedestrian detection.

(a) HOG (b) R_p+HOG (c) HOG (d) R_p+HOG

Figure 5.3: Sample outputs of HOG and $R_p + HOG$ on MIT dataset.

5.2.2 Combining Information from Magno Channel and Parvo Channel

The Magno channel of retina is useful in extracting motion information from the images. In Fig. 5.4, sub-figs. 5.4a and 5.4b show the unprocessed image

Table 5.2: Performance measures from INRIA Dataset

	HOG	$R_p + HOG$	C^4	$R_p + C^4$
FN	138	144	148	144
FP	220	251	1	0
TP	994	988	984	988
FPR	0.18122	0.20258	0.00102	0
FNR	0.12191	0.12721	0.13074	0.12721
DR	0.87809	0.87279	0.86926	0.87279
DP	0.81878	0.79742	0.99898	1

Table 5.3: Performance measures from Caltech Dataset

	HOG	$R_p + HOG$	C^4	$R_p + C^4$
FN	1361	1099	1304	925
FP	113	141	146	144
TP	4048	4310	4105	4484
FPR	0.02715	0.03167	0.03434	0.03111
FNR	0.2516	0.2031	0.2410	0.1710
DR	0.7483	0.7968	0.7589	0.8289
DP	0.9728	0.9683	0.9656	0.9688

and output from magno channel respectively. For an image i, let C_j^i denote the j^{th} contour identified in image i where j is varied from zero to total number of contours identified (n). Algorithm proposed by Suzuki et al [126] is utilized for contour extraction. The extracted contours are filtered based on the contour area. Let A_j^i denote the area of the contour j from image i and A_{thresh} denote the minimum area of contour below which the contours will be discarded. Filtering contours is necessary to limit unwanted processing. If a pedestrian is present in the image and if there is movement from the pedestrian, it is noted that the contour area is always above the A_{thresh}. These contours are compared with the output of $R_p + HOG$ or $R_P + C^4$ to identify the pedestrians in the image. If an output of the pedestrian detector from Parvo channel and contour from Magno channel coincide then, that output ($P^i_{(x,y)}$) is confirmed as a pedestrian. Algorithm 5.1 explains the overall process for pedestrian detection utilizing a retina model and Fig. 5.5 is a pictorial representation of the overall process.

(a) Unprocessed image (b) Ouput of Magno channel

Figure 5.4: Images processed by Magno Channel

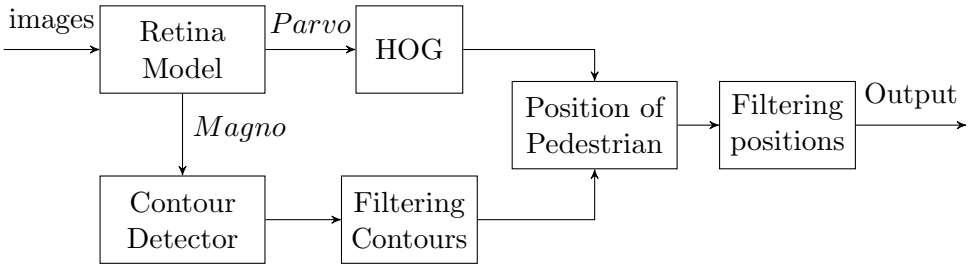


Figure 5.5: Overview of pedestrian process combining output from Parvo and Magno channel of retina.

5.2.3 Utilizing Temporal Information for Filtering Positions of Pedestrians Detected

The position of the pedestrian detected $P^i_{(x,y)}$, for an image i may not be correct always. These false positives can be identified by utilizing temporal information. A tracklet $T_{P_{(x,y)}}$, is utilized to store past positions of pedestrians detected to study the temporal information. Tracklet $T_{P_{(x,y)}}$ stores positions of the pedestrians identified for the past two seconds. To make sure only information from the recent two seconds are utilized, older information are replaced with newer information. If m frames are obtained in two seconds, then $T_{P_{(x,y)}}$ contains the $P_{(x,y)}$ information of pedestrians identified over m image frames. Euclidean distance, direction and velocity of the pedestrians detected are utilized to identify the outliers. Based on the information on $T_{P_{(x,y)}}$, thresholds for each of the outlier detector (threshold for Euclidean distance (Δe), threshold for direction ($\Delta \bar{d}$) and threshold for velocity (Δv) of the pedestrian detected) are decided. The output of outlier detectors is either 1 or 0. The outlier thresholds are updated as $T_{P_{(x,y)}}$ is updated (after each window). Under the assumption that only one pedestrian is present in each

Algorithm 5.1 Algorithm for Retina-model-based Pedestrian Detection

Input: image i at time t, Threshold for contour area A_{thresh} and tracklet $T_{P_{(x,y)}}$ containing positions of pedestrians detected for the past m image frames $P^j_{(x,y)}, j = 1, ..m$.

Apply retina filter to the input image i to get Parvo output R_p and Magno Output R_m.
Apply HOG or C^4 to the Parvo output $R_p + HOG$ or $R_p + C^4$.
Find all contours (C^i_j) in R_m.
Compute area (A^i_j) of all C^i_j.
for All contours (C^i_j) **do**
  if $A^i_j \geq A^i_{thresh}$ **then**
    $C^i_{j,filtered} = C^i_j$
  end if
end for
for All filtered contours $(C^i_{j,filtered})$ **do**
  if $C^i_{j,filtered} \cap R_p + HOG$ (If HOG is utilized as the standard detector) **then**
    Position of pedestrian detected $P^i_{(x,y)}$ = centroid of $C^i_{j,filtered} \cap R_p + HOG$.
  end if
end for
if $||P^i_{(x,y)} - P^j_{(x,y)}|| \leq \delta e$ **then**
  $E_{out} = 1$
end if
if $tan^{-1}(P^i_{(x,y)}, P^j_{(x,y)}) \leq \delta \bar{d}$ **then**
  $\bar{D}_{out} = 1$
end if
if $\frac{||P^i_{(x,y)} - P^j_{(x,y)}||}{\Delta t} \leq \delta v, \Delta t = t_i - t_j$ **then**
  $V_{out} = 1$
end if
if E_{out} and $\bar{D}_{out}$ and $V_{out} = 1$ **then**
  Final position of the pedestrian detected = $P^i_{(x,y)}$
  add $P^i_{(x,y)}$ to tracklet $T_{P_{(x,y)}}$
end if

image frame, if $P^i_{(x,y)}$ is the position of the pedestrian detected in the current image frame, the outlier detection is performed as follows,

$$E_{out} = \begin{cases} 1 & \text{if } ||P^i_{(x,y)} - P^j_{(x,y)}|| \leq \Delta e, \\ 0 & \text{otherwise, } P^j_{(x,y)} \in T_{P_{(x,y)}}, j = 1, ..m \end{cases} \tag{5.7}$$

$$\bar{D}_{out} = \begin{cases} 1 & \text{if } tan^{-1}(P^i_{(x,y)}, P^j_{(x,y)}) \leq \Delta \bar{d}, \\ 0 & \text{otherwise, } P^j_{(x,y)} \in T_{P_{(x,y)}}, j = 1, ..m \end{cases} \tag{5.8}$$

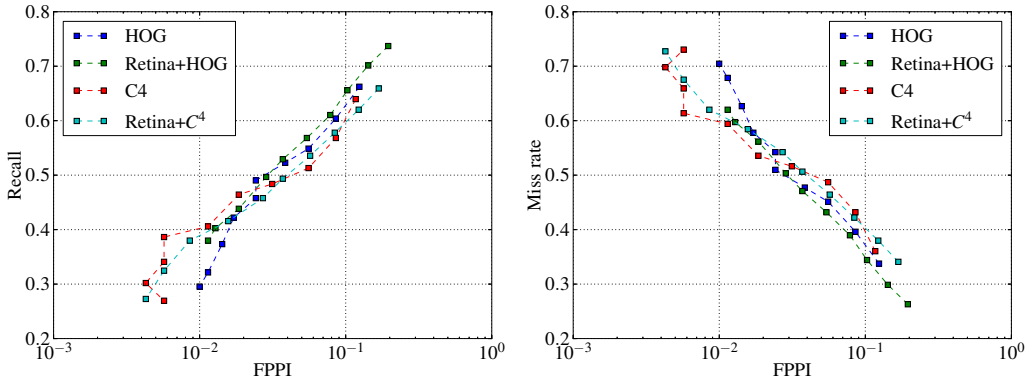
$$V_{out} = \begin{cases} 1 & \text{if } \frac{||P^i_{(x,y)} - P^j_{(x,y)}||}{\Delta t} \leq \Delta v, \Delta t = t_i - t_j \\ 0 & \text{otherwise, } P^j_{(x,y)} \in T_{P_{(x,y)}}, j = 1, ..m \end{cases} \tag{5.9}$$

E_{out}, $\bar{D}_{out}$ and V_{out} are the outputs of Euclidean distance-based, direction-based and velocity-based threshold detectors. A pedestrian is considered detected only if outputs of all the threshold detectors (E_{out}, $\bar{D}_{out}$ and V_{out}) are 1.

5.3 Performance Analysis of Human Visual Cortex Model-based Pedestrian Detectors

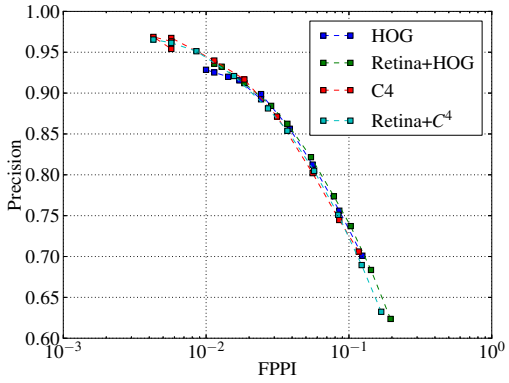
Performance of standard pedestrian detectors HOG and C^4 pedestrian detectors are compared with retina-model-based HOG (Retina+HOG) and retina-model-based C^4 (Retina+C^4) pedestrian detectors. Caltech [28] and Daimler [31] pedestrian benchmark datasets are utilized for the performance analysis. In both Caltech and Daimler pedestrian datasets, scenarios such as a pedestrian walking towards a stationary camera, camera moving towards a stationary pedestrian and both pedestrian and camera moving towards or away from each other, are included for analysis. True positive rate (recall), false negative rate (miss rate), positive predictive value (precision) and True negative rate (specificity) are utilized as performance measures for comparing the pedestrian detectors. The performance measures are plotted against False Positive Per Image (FPPI) to analyse the performance of the detectors across different thresholds [132]. Figs. 5.6 and 5.7 shows the performance of HOG, C^4, Retina+HOG and Retina+C^4 pedestrian detectors for Caltech and Daimler pedestrian benchmark datasets respectively.

True positive rate or recall is a statistical measure of performance that measures the proportion of positives that are correctly identified. Higher the recall values, better the performance of the detector. Figs. 5.6a and 5.7a show the plot of true positive rate vs false positive per image for Caltech and Daimler pedestrian benchmark datasets respectively. In both the plots (Figs. 5.6a and 5.7a), it can be seen that retina-model-based detectors outperform standard detectors. In both the datasets, the highest recall value of HOG is lower than that of Retina+HOG while the lowest recall value of HOG is significantly lower than that of Retina+HOG. Similarly, the highest and lowest recall value of Retina+C^4 is significantly higher than that of C^4 in Daimler dataset while the performances

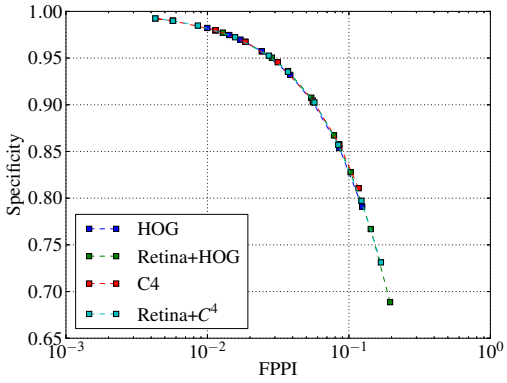


(a) Recall Vs FPPI

(b) Miss rate vs FPPI



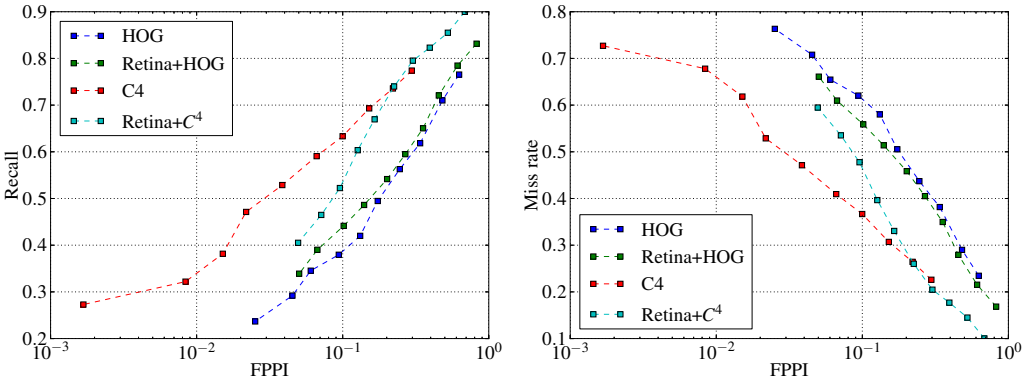
(c) Precision Vs FPPI



(d) Specificity vs FPPI

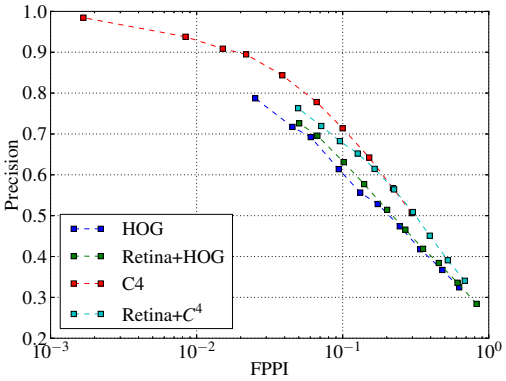
Figure 5.6: Performance curves for Caltech Dataset

are comparable in Caltech dataset. False negative rate or miss rate is a statistical measure of performance that measures the proportion of positives that are not correctly identified. Lower the miss rate, better the performance of the detector. Figs. 5.6b and 5.7b show the plot of miss rate vs false positive per image for Caltech and Daimler pedestrian benchmark datasets respectively. In both the datasets, retina-model-based detectors (Retina+HOG and Retina+C^4) have lower miss rates compared to standard HOG and C^4 detectors. The lowest and highest miss rate of Retina+HOG is lower than that of HOG in both the datasets. Similarly Retina+C^4 outperforms C^4 in both Daimler and Caltech datasets. Figs. 5.6c, 5.7c and Figs. 5.6d, 5.7d show the plots of precision and specificity against FPPI in Caltech and Daimler benchmark datasets respectively. These plots indicate that the precision and specificity of all the detectors are comparable. This study proves that the performance of human visual cortex-

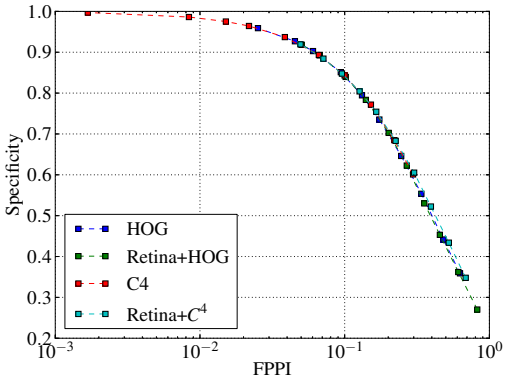


(a) Recall Vs FPPI

(b) Miss rate vs FPPI



(c) Precision Vs FPPI



(d) Specificity vs FPPI

Figure 5.7: Performance curves for Daimler Dataset

based pedestrian detectors are better in many situations and comparable in the rest of the situations.

Retina+HOG's maximum recall value is greater than 0.7 and 0.8 in Caltech and Daimler datasets while Retina+C^4's is greater than 0.6 and 0.9 for Caltech and Daimler datasets. While the maximum recall of Retina+C^4 is greater than Retina+HOG in Daimler dataset, its lowest recall value is significantly lower than Retina+HOG in Caltech dataset. Similarly maximum miss rate for Retina+C^4 is significantly higher than Retina+HOG in Caltech dataset while in Daimler dataset the maximum and minimum miss rates of Retina+C^4 and Retina+HOG are comparable. Specificity and precision of Retina+HOG and Retina+C^4 are comparable in both the datasets. As the performance of Retina+HOG is consistent across datasets, Retina+HOG is incorporated into SOIFRA and utilized for real-time pedestrian detection experiments.

5.4 Incorporating Pedestrian Detection and Avoidance Behaviour into SOIFRA

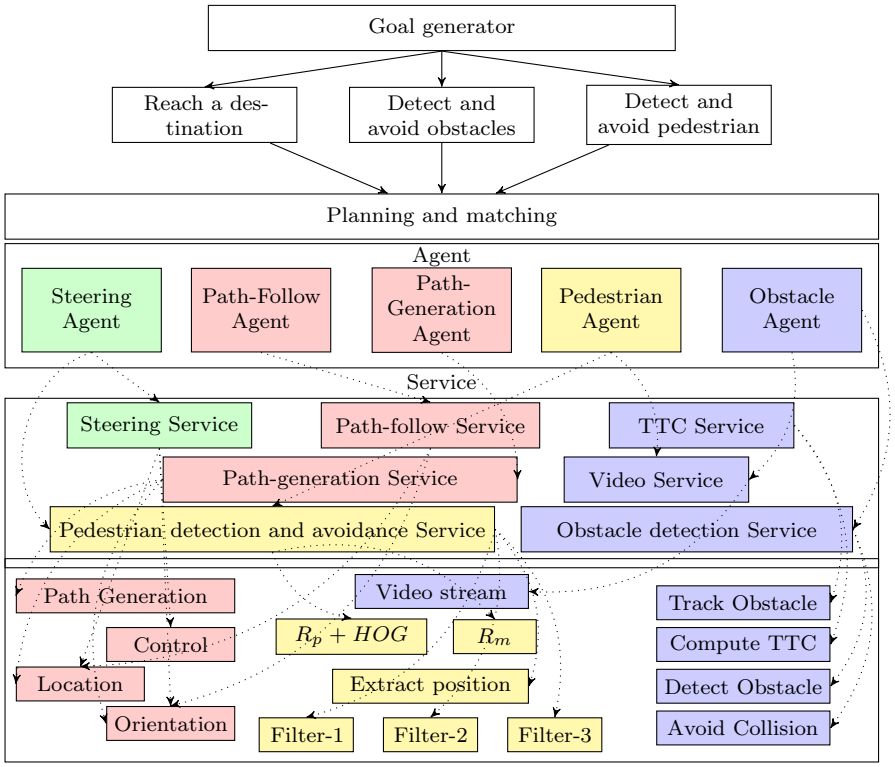


Figure 5.8: Architectural overview of SOIFRA with collision avoidance, path-generation, path-following and pedestrian detection and avoidance.

The architecture of SOIFRA, a multi-agent framework focused on achieving robot autonomy across UAVs and UGVs, is explained in detail in Section. 2.2 of Chapter 2. SOIFRA utilizes platform independent algorithms to achieve robot autonomy. Incorporation of collision avoidance and path-generation and path-following ability into SOIFRA is discussed in Section. 3.2 of Chapter 3 and Section. 4.3 of Chapter 4 respectively. This section explains the incorporation of Retina+HOG for pedestrian detection and pedestrian avoidance behaviour into SOIFRA. SOIFRA is a behavior based multi-agent framework made up of deliberation, behavior and execution layers. The robots are required to reach a specific destination in an unknown environment, avoiding pedestrians. The goal generator module splits the mission into three sub-goals; a sub-goal to make the robot reach a destination, a sub-goal to detect and avoid collisions and a sub-goal

to detect and avoid pedestrians. The planner-matcher module allocates agents to sub-goals based on pre-defined plans retrieved from ontological database in accordance with the prevailing states as explained in Section. 2.2. Fig. 5.8 shows the structure of SOIFRA adopted for real-time pedestrian detection experiments.

This mission is undertaken to demonstrate the pedestrian detection and avoidance capability of SOIFRA. The framework discussed in Section. 4.3 is enhanced to include the pedestrian detection and avoidance capability. The steering agent (Str:AGT) navigating the robot, obstacle agent (Obs:AGT) performing obstacle detection and avoidance, path-generation agent (PG:AGT) generating paths and path-follow agent (PF:AGT) providing path-following ability are carried over into this framework. The services and actions performed by the above mentioned agents are explained in Section. 3.2 and Section. 4.3. In order to provide pedestrian detection and avoidance capability a new agent called pedestrian agent (Ped:AGT) is developed. Ped:AGT provides the pedestrian detection and avoidance service (pda:SRV). Ped:AGT provides pedestrian detection ability utilizing the action to detect pedestrians (pdet:ACT). pdet:ACT utilizes Retina+HOG to detect pedestrians. The action to detect pedestrian (pdet:ACT) is a complex action made up of several small actions such as actions to perform $R_p + HOG$, R_m, extract position of the pedestrian and obtain the final position of the pedestrian after filtering. These actions combine to form pdet:ACT. Path-generation and Path-follow agents PG:AGT and PF:ACT utilize VDPGT to generate and follow paths to reach the desired destination. Obstacle agent Obs:AGT initiates the video service (video:SRV) and TTC service (TTC:SRV) to detect and avoid obstacles. TTC:SRV utilizes TTC-EO (explained in Section. 3.1.2.3) to estimate the distance to an obstacle detected.

Obs:AGT, PF:AGT and Ped:AGT publish control velocities for the robot to follow on *rostopic*. Upon initialization the priority index of all the collaborative agents are assigned. As pedestrian detection and avoidance is very crucial Ped:AGT is assigned the highest priority by default while Obs:AGT is assigned the second highest priority. PF:AGT is assigned the lowest priority. Str:AGT subscribes to all the three control velocities published but only utilizes the control

velocity from the agent with highest priority. Str:AGT utilizes the control velocity from PF:AGT if there are no obstacles or pedestrians detected in the image. Once a pedestrian is detected in the robot's navigation path, Ped:AGT estimates the position of the pedestrian detected in pixel coordinates. If the location of the pedestrian detected is near the optical center of the camera, Ped:AGT is assigned the highest priority and pedestrian avoidance is carried out. Unlike collision avoidance explained in Sections. 3.2 and 4.3, the target destination is changed 0.5m in Y-axis so that the robot does not enter the space of the pedestrian. Direction of translation (positive Y or negative Y) is decided based on the direction of travel of the pedestrian. Once the robot undergoes 0.5m translation along Y-axis, the PF:AGT attains higher priority, a new path to the new destination is generated and path-following is carried out. Fig.5.9 shows the operational sequences for collaboration between path-follow and pedestrian detection agents. The collision avoidance process exactly the same as explained in Section.4.3.

5.5 Real-time Experimental Results Demonstrating Pedestrian Detection and Avoidance using SOIFRA

Real-time experiments are carried out utilizing AR Drone and Turtlebot. Both the robots utilize Retina+HOG to detect and avoid pedestrians in their path. Experiments are carried out in a corridor type of environment. Three scenarios are considered: scenario 1, where the robot is moving and the human is stationary. Three use cases for scenario 1 are explored as follows

1. the robots are in collision course with the pedestrian,
2. the pedestrian is to the left of the robot and
3. the pedestrian is to the right of the robot.

Scenario 2, where a pedestrian walks from outside the robot's visual region to the front of the moving robot. Scenario 3 is a continuation from from scenario 2

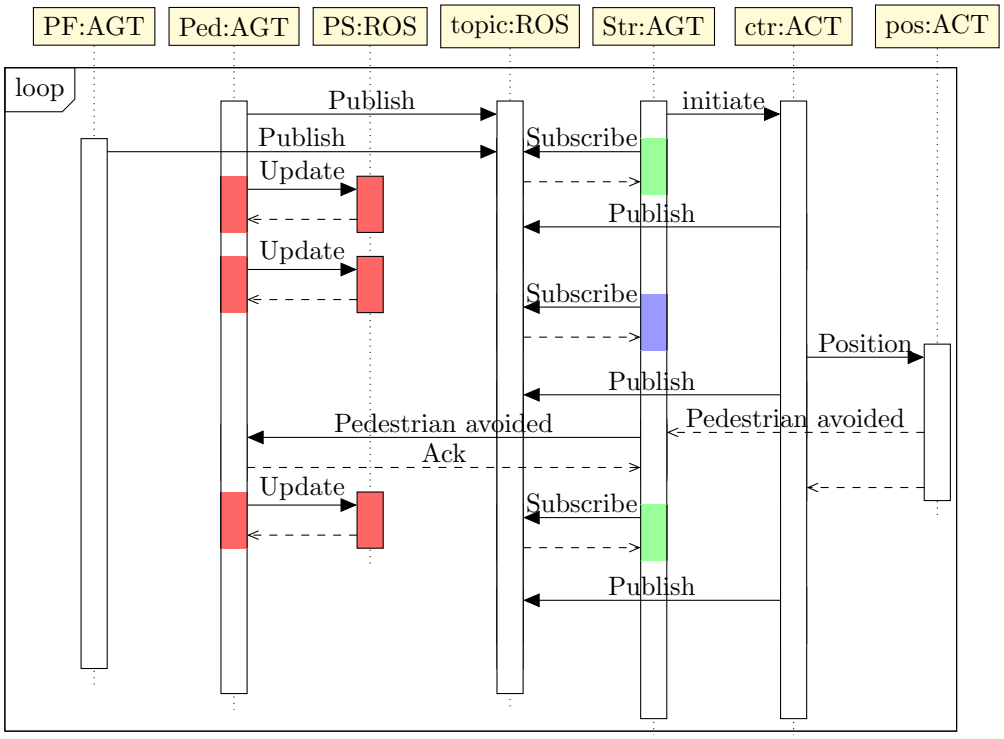


Figure 5.9: Operational sequences demonstrating collaboration between path-follow agent, PF:AGT and pedestrian-detection agent, Ped:AGT. AGT and ACT represent agent and action respectively. The Steering agent Str:AGT receives inputs from PF:AGT and Ped:AGT simultaneously through *rostopic*. When there is no pedestrian detected along the robot path, PF:AGT has higher priority. Once the robot is detected to be in a collision course with a pedestrian, Str:AGT assigns the highest priority to pedestrian avoidance. Once the pedestrian is avoided, PF:AGT regains higher priority.

where the robot as well as the pedestrian is moving towards each other after the pedestrian walks in-front of the robot. Figure 5.10 gives a pictorial representation of the position of pedestrians for different scenarios. A is the starting point for the robots and B is the target location. For scenario 1, pedestrians are at C-1, C-2 and C-3 respectively. For scenario 2, the pedestrian is initially out of the visual region. The human then appears at C-4 and moves towards C5. For scenario 3, the pedestrian is out of the viewing region of the robot initially, and moves towards C-2, crossing C-4 and C-5 and continues to move towards A, while the robot is moving from A to B.

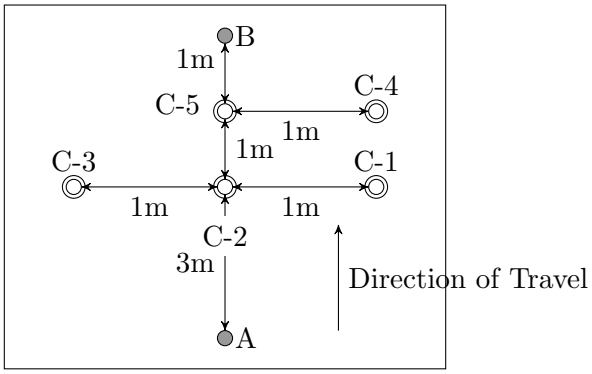
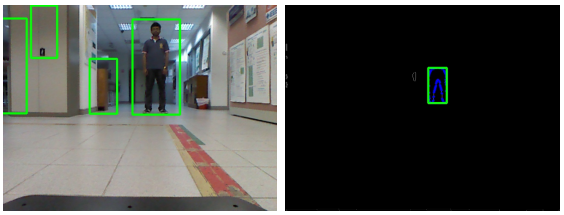


Figure 5.10: Layout showing the position of humans in the different scenarios of the real-time experiments. The robots start from A, move towards and stop at B.

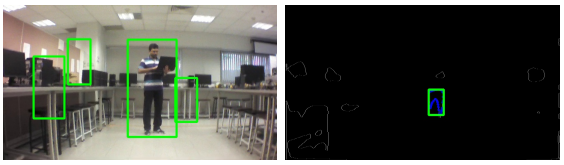
5.5.1 Scenario – 1

In scenario 1, the robots are moving while the pedestrian is stationary. C-1, C-2 and C-3 in Fig. 5.10 are the positions of stationary pedestrians while A and B represent the starting and target locations of the robots. Experiments for the three cases are carried out separately. Velocity of Turtlebot is fixed at 0.1m/s while the AR Drone is operated at 0.65m/s. All the agents, path-planning (PP:AGT), path-Follow (PF:AGT), pedestrian (Ped:AGT), Obstacle (Obs:AGT) and Steering (Str:AGT) agents are initiated at the same time. PP:AGT generates the shortest path to reach B from A utilizing VDPGT algorithm and conveys this path to PF:AGT thorough *rostopic*. PF:AGT publishes the control parameters required for path-tracking to the Str:AGT. At the same time Ped:AGT, utilizing pdet:ACT, estimates the location of the pedestrian. If the pedestrian is around the centre of the image (nearer to the optical center), Str:AGT gives higher priority to pedestrian avoidance, otherwise path-following attains higher priority. For case-1 and case-3 where pedestrians are at C-1 and C-3, respectively, the pedestrian avoidance should not be executed while for case-2 (C-2), pedestrian avoidance is to be carried out before path-following sequence. Fig. 5.11a and 5.11b show a sample output using HOG and Retina+HOG detectors from a Turtlebot and AR Drone respectively. It can be seen from Figs. 5.11a and 5.11b that Retina+HOG is better in eliminating false positives and accurately detecting the pedestrian in the image. Fig. 5.12b shows the positions of pedestrians detected

using Retina+HOG, throughout the real-time experiments for scenario-1. It can be seen from Fig. 5.12b that for case-1 and case-3, the position of pedestrians estimated is not in the path-of-travel of the robot. Fig. 5.12a shows the path travelled by both Turtlebot and AR Drone for case-2 of scenario-1. For case-2, the Ped:AGT identifies that the robot is in collision course with the detected pedestrian and it initiates the avoidance sequence immediately.



(a) Real-time experiments on Tutlebot.

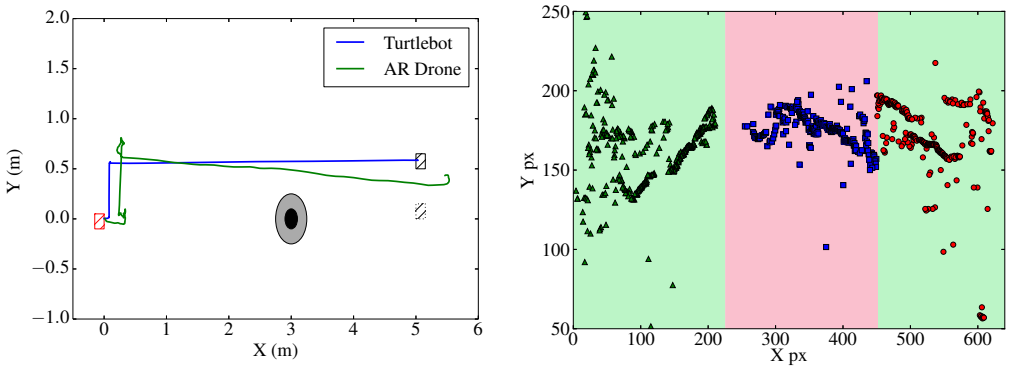


(b) Real-time experiments on AR Drone.

Figure 5.11: Real-time experiments with Turtlebot and AR Drone for scenario 1 (the pedestrian is stationary while the robot is moving). The left images are pedestrian detection outputs using HOG while right images are outputs using Retina+HOG.

5.5.2 Scenario – 2

In Scenario 2, the pedestrian is initially out of the field of view of the robots. The robots start moving to B from A. After some time, the pedestrian appears at C-4 and continues to walk towards C-5 while the robot is moving to B from A. Pedestrian detection agent (Ped:AGT) continually tracks the pedestrian's movement from C-4 to C-5. Once Ped:AGT identifies that the robot is in collision course with the pedestrian, pedestrian avoidance is carried out. Since the pedestrian is moving from C-4 to C-5 the pedestrian avoidance process creates the new destination by moving the current destination by 0.5m along positive Y-axis. Fig. 5.13a and 5.13b show the a sample output using HOG and Retina+HOG detectors for scenario 2 for a Turtlebot and AR Drone respectively. Fig. 5.14b shows the positions of pedestrians detected using Retina+HOG, throughout

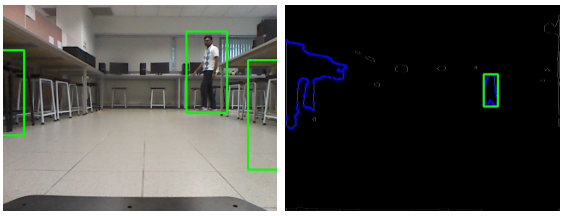


(a) Path travelled by the robots. The ellipse represents the human (top view)

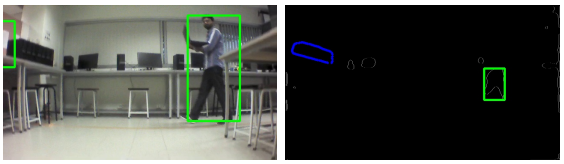
(b) Position of the pedestrian detected and tracked throughout the experiments.

Figure 5.12: Path travelled by Turtlebot and AR Drone during the real-time experiments and positions of pedestrian detected utilizing Retina+HOG for scenario–1. In subfig (b), the regions where the robot does not perform pedestrian avoidance are shown in green (for the cases where the pedestrian is to the left/right of the robot). The robots perform pedestrian avoidance if there are pedestrians detected in the red region.

the real-time experiments for scenario-2. It can be seen from Fig. 5.14b that Retina+HOG is able to detect the pedestrians as he moves from C-4 to C-5. Fig. 5.14a shows the path travelled by Turtlebot and AR Drone. It is seen from Fig. 5.14a that both AR Drone and Turtlebot modify their desired destination and successfully avoid the pedestrian.

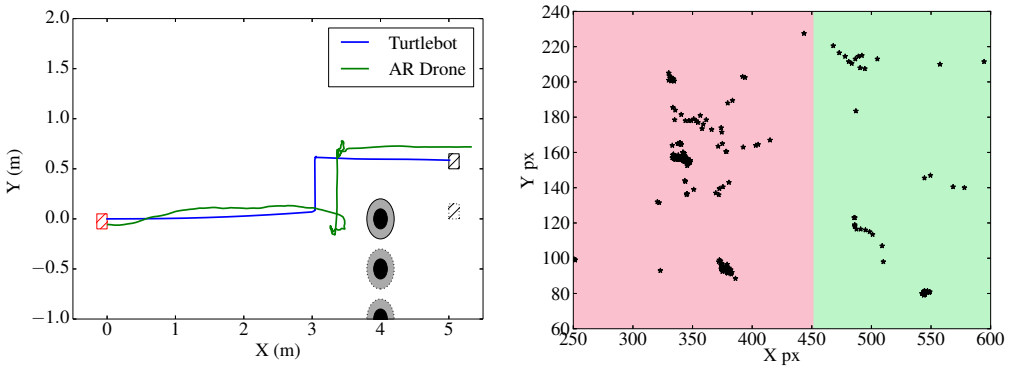


(a) Real-time experiments on Turtlebot.



(b) Real-time experiments on AR Drone.

Figure 5.13: Real-time experiments with Turtlebot and AR Drone for scenario 2 (the robots are moving to B from A, while the pedestrian enters the scene at C-4 and walks towards C-5). The left images are pedestrian detection outputs using HOG while right images are outputs using Retina+HOG.



(a) Path travelled by the robots. The ellipse represents the human (top view)

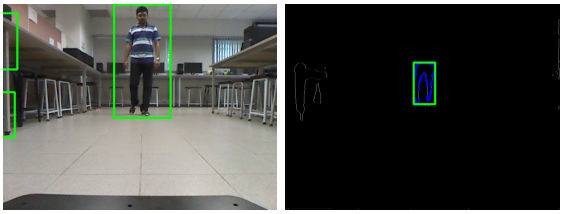
(b) Position of the pedestrian detected and tracked throughout the experiments.

Figure 5.14: Path travelled by Turtlebot and AR Drone during the real-time experiments and positions of pedestrian detected utilizing Retina+HOG for scenario–2. The robots perform pedestrian avoidance if the position of the detected pedestrians is in the red region of subfig (b).

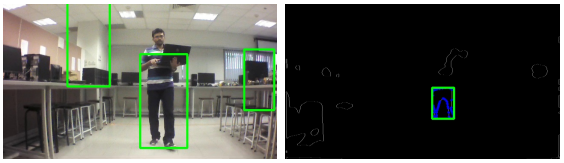
5.5.3 Scenario – 3

Scenario 3 is similar to scenario 2, the difference being the pedestrian continuing to move towards C-2 after reaching C-5. It can be seen from Fig. 5.16b that the robots track the human as the human moves from C-4 to C-2 and takes evasive action after the robot decides that the robot is in collision coarse with the human. Fig. 5.15a and 5.15b show a sample output using HOG and Retina+HOG detectors for a Turtlebot and AR Drone respectively as the human continues to walk towards A from C-2. It can be seen from Figs. 5.16b and Figs. 5.16a that Retina+HOG is able to detect pedestrians who are moving and successfully avoid collision with the human. Fig. 5.17 compares the output from HOG and Retina+HOG pedestrian detectors when there is no human in the scene.

Retina+HOG pedestrian detector manages to achieve relatively less false detections compared to standard HOG in all the three scenarios. Combining the motion information from magno channel with $R_p + HOG$ output helps to eliminate most of the possible false positives. In addition to this, the locally enhanced contour information provided by parvo channel improves HOG's detection performance in scenarios where the images are less clear (for example images in MIT dataset). Even though the computational load is more compared

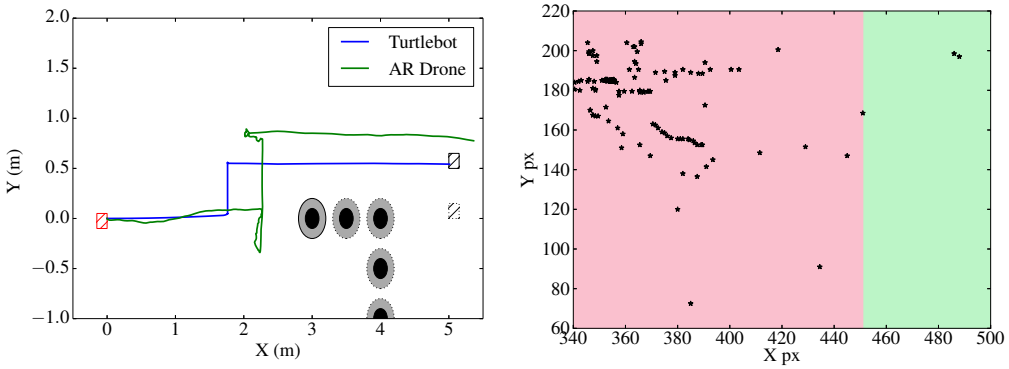


(a) Real-time experiments on Turtlebot.



(b) Real-time experiments on AR Drone.

Figure 5.15: Real-time experiments with Turtlebot and AR Drone for scenario 3 (the robots are moving to B from A, while the pedestrian enters the scene at C-4, walks towards C-2 crossing C-5). The left images are pedestrian detection outputs using HOG while right images are outputs using Retina+HOG.



(a) Path travelled by the robots. The ellipse represents the human (top view)

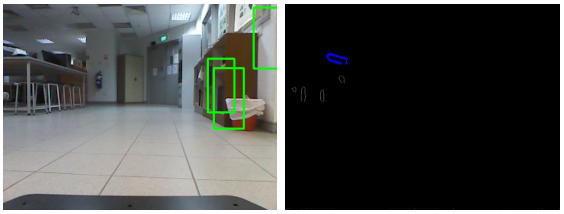
(b) Position of the pedestrian detected and tracked throughout the experiment.

Figure 5.16: Path travelled by Turtlebot and AR Drone during the real-time experiments and positions of pedestrian detected utilizing Retina+HOG for scenario–3. The robots perform pedestrian avoidance if the position of the detected pedestrians is in the red region of subfig (b).

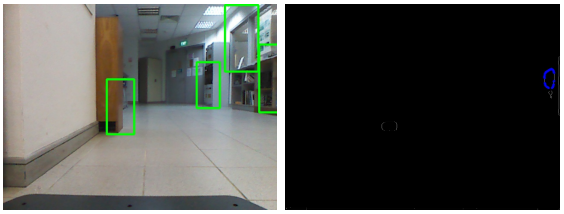
to standard HOG or C^4 detectors, the improved performance of Retina+HOG is very helpful in real-time environments where the detection accuracy is crucial.

5.6 Summary

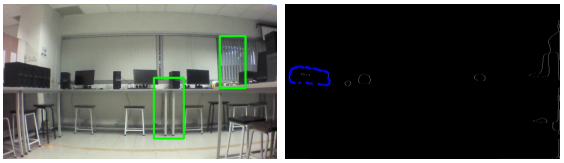
This chapter presents a human visual system-based model that is combined with state-of-the-art pedestrian detection algorithms to achieve better pedestrian detection accuracy. Retina model is of use when contours and image contrasts



(a) Sample-1



(b) Sample-2



(c) Sample-3

Figure 5.17: HOG and $R_p + HOG$ output from negative samples during the real-time experiments. The black images are the output from $R_p + HOG$. It can be seen clearly that $R_p + HOG$ correctly identifies the negative samples while HOG produces multiple false positives from the same samples.

are of importance. The Parvo channel of retina, useful in extracting locally enhanced contours or other such information, when combined with a standard pedestrian detector such as HOG or C^4 detectors achieve comparatively better performances than standard HOG and C^4 detectors. The pedestrian detection performances from INRIA, MIT and Caltech pedestrian benchmark datasets helps to establish the performance improvement of $R_p + HOG$ and $R_P + C^4$ pedestrian detectors. The magno channel of retina, useful in identifying motion information, when combined with the pedestrian detection information from the parvo channel, the detection accuracy is improved significantly. Recall, miss rate, precision and specificity vs FPPI plots of HOG, C^4, Retina+HOG and Retina+C^4 obtained from Caltech and Daimler pedestrian benchmark datasets support that the combination of retina model with a standard pedestrian detector provides improved performance. SOIFRA, an interoperable framework for robot autonomy, is modified to accommodate Retina+HOG in addition to collision avoidance,

path-generation and following algorithms. A pedestrian agent that provides pedestrian detection and avoidance capability is incorporated into SOIFRA. The pedestrian detection agent utilizes Retina+HOG, a human visual cortex-based pedestrian detector, for real-time pedestrian detection and tracking. Experiments conducted utilizing AR Drone and Turtlebot also demonstrate that Retina+HOG performs well in real-time scenarios. Real-time experiments are conducted for three scenarios: a scenario where the pedestrian is stationary while the robot is moving, a scenario where the pedestrian as well as robot is moving and a scenario where the pedestrian enters the scene and moves towards a moving robot. In all the scenarios the robots using Retina+HOG are successful in detecting and avoiding the pedestrians in real-time.

Chapter 6

Conclusion & Directions for Future Research

"Everyone who got where he is has had to begin where he was"

– Napolean Hill

6.1 Conclusions

The primary aim of this thesis is to develop a multi-agent framework that allows ease of usage of both UAVs and UGVs for the same missions, with minimum modifications. SOIFRA framework proposed in Chapter 2 is an interoperable behaviour-based multi-agent framework that accommodates platform independent algorithms. SOIFRA is a behaviour-based framework and several behaviours that are fundamental for autonomous operation of UAVs and UGVs are incorporated into SOIFRA. Collision avoidance, basic requirement for every autonomous robot is incorporated into SOIFRA as explained in Chapter 3. Vision sensors are utilized for obstacle detection in SOIFRA. Obstacle avoidance is carried out by estimating the TTC an obstacle detected. Two separate experiments with an optical flow-based TTC method and an expansion of objects-based TTC are performed, demonstrates the modularity of SOIFRA. A mission where the robots are required to reach a target in an unknown environment by avoiding unknown

obstacles carried-out successfully, demonstrates the collision avoidance capability of SOIFRA.

Real-time path-generation and path-following ability is also incorporated into SOIFRA as discussed in Chapter 4. VDPGT, a geometry-based real-time path-generation and path-following algorithm developed, is incorporated into SOIFRA. Simulation and real-time experiments conducted on aerial and ground robots separately, successfully demonstrate the performance and platform independent nature of VDPGT. A mission, that requires the aerial and ground robots to collaborate and navigate a GPS denied urban environment utilizing VDPGT, portrays the usability of VDPGT in realistic missions. Another mission that require the robot to reach a target destination in an unknown environment with obstacles successfully demonstrate the real-time path-generation and path-following ability of SOIFRA.

The pedestrian detection algorithm explained in Chapter 5 is a human visual cortex model-based pedestrian detection algorithm. This pedestrian detection algorithm is incorporated into SOIFRA to add human collision avoidance behaviour. Mathematical models of Parvo and Magno channel of human retina combined with existing state-of-the art pedestrian detectors successfully improve the pedestrian detection accuracy. The improvement in pedestrian detection accuracy is validated through several experiments on existing pedestrian benchmark datasets. The performance of the proposed pedestrian detection algorithm incorporated into SOIFRA is demonstrated through a series of real-time experiments where the robots successfully detect and avoid stationary as well as moving pedestrians.

The results from simulation and real-time experiments discussed in Chapter 3, Chapter 4 and Chapter 5 show that SOIFRA is capable of collision avoidance, real-time path-generation and path-following and pedestrian detection abilities. The results also prove that SOIFRA is an interoperable framework that accommodates algorithms that work independent of the platforms utilized.

6.2 Directions for Future Research

Interoperable or common multi-agent frameworks for robotic systems have been relatively less explored. This thesis aimed to solve this by developing SOIFRA framework to provide a common framework for UAVs and UGVs. However there are other types of robotics systems such as under water robots, unmanned surface robots and space robots. Further studies could focus on including other robotic systems into SOIFRA framework to extend SOIFRA's interoperability.

SOIFRA mainly focuses on camera and vision-based algorithms to attain platform independence. Further exploration on other sensors that provide platform independence can also be looked into.

Collision avoidance, path-generation and path-following and pedestrian detection behaviours are included into SOIFRA. The collision avoidance algorithm incorporated into SOIFRA is useful in environments where the obstacles have definite edges. This algorithm can be further improved to include detection mechanisms that detects all types of obstacles.

Even though the shortest path to a destination is always a straight line, in certain scenarios following a curved trajectory could be the optimal path. Hence VDPGT could be improved by including the capacity to generate curved trajectories in specific situations.

The retina-model-based pedestrian detection algorithm developed can successfully detect multiple pedestrians. However it cannot track multiple pedestrians in the scene. This is also a possible research direction that can be explored.

At present only basic behaviours such as collision avoidance, path-following and pedestrian detection are incorporated into SOIFRA. More complex behaviours such as patrolling, foraging and evading, essential for other missions, can be incorporated into SOIFRA. SOIFRA can be modified to accomodate multiple robots at the same time (multi-robot system). At the present stage SOIFRA is capable of accomodating only one robot. There can be only one obstacle agent operating simultaneously. If there are more than one, same type agent operating at the same time, then SOIFRA may face deadlock or livelock condition.

Publications Resulting from this Thesis

Published journal papers

1. W.A. Arokiasami, P. Vadakkepat, K.C. Tan and D. Srinivasan, Interoperable multi-agent framework for unmanned aerial/ground vehicles: towards robot autonomy, *Complex & Intelligent Systems*, vol.2, no.1, pp. 45–59, 2016.

Submitted journal papers

2. W.A. Arokiasami, P. Vadakkepat, K.C. Tan and D. Srinivasan (2016). Interoperable Multi-Agent framework for Unmanned Aerial/Ground Vehicles with Real-time Path-Generation and Tracking, submitted to *Unmanned Systems*.
3. W.A. Arokiasami, P. Vadakkepat, K.C. Tan and D. Srinivasan (2016). Human Visual System based pedestrian Detection and Avoidance for Unmanned Aerial/Ground Vehicles using a Multi-agent Framework, submitted to *IEEE Transactions on Cybernetics*.

Conference papers

4. W.A. Arokiasami, K.C. Tan, D. Srinivasan and P. Vadakkepat (2015). Impact of the Length of Optical Flow Vectors in Estimating Time-to-contact an Obstacle. Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems, Volume 2, Springer International Publishing, pp. 201–213, 2015.
5. W. A. Arokiasami, P. Vadakkepat, K. C. Tan and D. Srinivasan, Vector directed path generation and tracking for autonomous unmanned aerial/ ground vehicles, 2016 IEEE Congress on Evolutionary Computation (CEC), Vancouver, BC, Canada, 2016, pp. 1375-1381.

Bibliography

[1] S. Akishita, S. Kawamura, and K. Hayashi. Laplace potential for moving obstacle avoidance and approach of a mobile robot. In *Japan-USA Symposium on flexible automation, A Pacific rim conference*, pages 139–142, 1990.

[2] G. Alenyà, A. Nègre, J. L. Crowley, et al. Time to contact for obstacle avoidance. 2009.

[3] I. P. Alonso, D. F. Llorca, M. Á. Sotelo, L. M. Bergasa, P. R. de Toro, J. Nuevo, M. Ocaña, and M. Á. G. Garrido. Combination of feature extraction methods for svm pedestrian detection. *IEEE Transactions on Intelligent Transportation Systems*, 8(2):292–307, 2007.

[4] G. Ambrosino, M. Ariola, U. Ciniglio, F. Corraro, E. De Lellis, and A. Pironti. Path generation and tracking in 3-d for uavs. *Control Systems Technology, IEEE Transactions on*, 17(4):980–988, 2009.

[5] R. C. Arkin. Towards cosmopolitan robots: Intelligent navigation in extended man-made environments. 1987.

[6] H. Asama, A. Matsumoto, and Y. Ishida. Design of an autonomous and distributed robot system: Actress. In *IEEE/RSJ IROS*, pages 283–290, 1989.

[7] B. I. Badano. *A multi-agent architecture with distributed coordination for an autonomous robot.* PhD thesis, Ph. D. thesis, Universitat de Girona, 2008.

[8] A. Baumberg. Hierarchical shape fitting using an iterated linear filter. *Image and vision computing*, 16(5):329–335, 1998.

[9] W. Beaudot. The neural information processing in the vertebrate retina: A melting pot of ideas for artificial vision. *Computer science, INPG, Grenoble*, 51, 1994.

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

[11] A. Benoit, A. Caplier, B. Durette, and J. Hérault. Using human visual system modeling for bio-inspired low level image processing. *Computer vision and Image understanding*, 114(7):758–773, 2010.

[12] V. Berenz, F. Tanaka, K. Suzuki, and M. Herink. Tdm: A software framework for elegant and rapid development of autonomous behaviors for humanoid robots. In *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, pages 179–186. IEEE, 2011.

[13] R. A. Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2(1):14–23, 1986.

[14] R. A. Brooks. Intelligence without representation. *Artificial intelligence*, 47(1):139–159, 1991.

[15] N. A. Browning. A neural circuit for robust time-to-contact estimation based on primate mst. *Neural computation*, 24(11):2946–2963, 2012.

[16] D. Busquets, C. Sierra, and R. L. De Màntaras. A multiagent approach to qualitative landmark-based navigation. *Autonomous Robots*, 15(2):129–154, 2003.

[17] D. W. Cho. Certainty grid representation for robot navigation by a bayesian method. *Robotica*, 8(02):159–165, 1990.

[18] G. Cielniak, A. Treptow, and T. Duckett. Quantitative performance evaluation of a people tracking system on a mobile robot. In *Proc. 2nd European Conference on Mobile Robots*, 2005.

[19] J. H. Connell. Sss: A hybrid architecture applied to robot navigation. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 2719–2724. IEEE, 1992.

[20] T. F. Cootes, S. Marsland, C. J. Twining, K. Smith, and C. J. Taylor. Groupwise diffeomorphic non-rigid registration for automatic model building. In *European conference on computer vision*, pages 316–327. Springer, 2004.

[21] J. A. Corrales Ramón, G. J. García Gómez, F. Torres Medina, and V. Perdereau. Cooperative tasks between humans and robots in industrial environments. 2012.

[22] R. C. Coulter. Implementation of the pure pursuit path tracking algorithm. Technical report, DTIC Document, 1992.

[23] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[24] P. Davidsson. *Autonomous agents and the concept of concepts*. Department of Computer Science, Box 118, 221 00 Lund, Sweden,, 1996.

[25] K. Dawson-Howe. *A practical introduction to computer vision with opencv*. John Wiley & Sons, 2014.

[26] P. Doherty, J. Kvarnstrom, M. Wzorek, P. Rudol, F. Heintz, and G. Conte. Hdrc3: A distributed hybriddeliberative/reactive architecture for unmanned aircraft systems. In *Handbook of Unmanned Aerial Vehicles*, pages 849–952. Springer, 2015.

[27] P. Dollár, S. Belongie, and P. Perona. The fastest pedestrian detector in the west. In *BMVC*, volume 2, page 7. Citeseer, 2010.

[28] P. Dollár, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: A benchmark. In *CVPR*, June 2009.

[29] R. R. dos Santos, S. d. F. P. Saramago, and V. Steffen Jr. Planejamento de trajetória de robô autônomo através do conceito de campo potencial. 2006.

[30] M. Enzweiler and D. M. Gavrila. A mixed generative-discriminative framework for pedestrian classification. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[31] M. Enzweiler and D. M. Gavrila. Monocular pedestrian detection: Survey and experiments. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(12):2179–2195, 2009.

[32] M. Enzweiler, P. Kanter, and D. M. Gavrila. Monocular pedestrian recognition using motion parallax. In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 792–797. IEEE, 2008.

[33] L. Fan, K.-K. Sung, and T.-K. Ng. Pedestrian registration in static images with unconstrained background. *Pattern Recognition*, 36(4):1019–1029, 2003.

[34] B. Faverjon. Obstacle avoidance using an octree in the configuration space of a manipulator. In *Robotics and Automation. Proceedings. 1984 IEEE International Conference on*, volume 1, pages 504–512. IEEE, 1984.

[35] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[36] J. Firby. Adaptive execution in complex dynamic domains. *Unpublished doctoral dissertation*, 1989.

[37] F. Flacco, T. Kröger, A. De Luca, and O. Khatib. A depth space approach to human-robot collision avoidance. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 338–345. IEEE, 2012.

[38] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[39] J. Fritsch, M. Kleinehagenbrock, S. Lang, T. Plötz, G. A. Fink, and G. Sagerer. Multi-modal anchoring for human–robot interaction. *Robotics and Autonomous Systems*, 43(2):133–147, 2003.

[40] T. Fukuda, S. Nakagawa, Y. Kawauchi, and M. Buss. Structure decision method for self organising robots based on cell structures-cebot. In *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, pages 695–700 vol.2, May 1989.

[41] E. Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *AAAi*, volume 1992, pages 809–815, 1992.

[42] D. M. Gavrila. A bayesian, exemplar-based approach to hierarchical shape matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(8):1408–1421, 2007.

[43] D. M. Gavrila and S. Munder. Multi-cue pedestrian detection and tracking from a moving vehicle. *International journal of computer vision*, 73(1):41–59, 2007.

[44] C. Goerzen, Z. Kong, and B. Mettler. A survey of motion planning algorithms from the perspective of autonomous uav guidance. *Journal of Intelligent and Robotic Systems*, 57(1-4):65–100, 2010.

[45] J. Guldner and V. I. Utkin. Stabilization of non-holonomic mobile robots using lyapunov functions for navigation and sliding mode control. In

Decision and Control, 1994., Proceedings of the 33rd IEEE Conference on, volume 3, pages 2967–2972. IEEE, 1994.

[46] T. Heap and D. Hogg. Wormholes in shape space: Tracking through discontinuous changes in shape. In *Computer Vision, 1998. Sixth International Conference on*, pages 344–349. IEEE, 1998.

[47] M. Hebert. Active and passive range sensing for robotics. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 102–110. IEEE, 2000.

[48] J. Hérault and B. Durette. Modeling visual perception for image processing. In *Computational and Ambient Intelligence*, pages 662–675. Springer, 2007.

[49] G. Holland, P. Webster, J. Curry, G. Tyrell, et al. The aerosonde robotic aircraft: A new paradigm for environmental observations. *Bulletin of the American Meteorological Society*, 82(5):889, 2001.

[50] H. C.-H. Hsu and A. Liu. A flexible architecture for navigation control of a mobile robot. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 37(3):310–318, 2007.

[51] http://gazebosim.org/wiki/Architecture. Gazebo architecture.

[52] Y. Huang, K. Huang, L. Wang, D. Tao, T. Tan, and X. Li. Enhanced biologically inspired model. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[53] S. Hwang, J. Park, N. Kim, Y. Choi, and I. So Kweon. Multispectral pedestrian detection: Benchmark dataset and baseline. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[54] Y. K. Hwang and N. Ahuja. A potential field approach to path planning. *IEEE Transactions on Robotics and Automation*, 8(1):23–32, 1992.

[55] C. C. Insaurralde and Y. R. Petillot. Capability-oriented robot architecture for maritime autonomy. *Robotics and Autonomous Systems*, 67:87–104, 2015.

[56] A. K. Jain, R. P. W. Duin, and J. Mao. Statistical pattern recognition: A review. *IEEE Transactions on pattern analysis and machine intelligence*, 22(1):4–37, 2000.

[57] N. R. Jennings. Coordination techniques for distributed artificial intelligence. *Foundations of distributed artificial intelligence*, pages 187–210, 1996.

[58] N. R. Jennings and K. Sycara. A roadmap of agent research and development. *Journal of Autonomous Agents and Multi-Agent Systems*, pages 275–306, 1998.

[59] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous agents and multi-agent systems*, 1(1):7–38, 1998.

[60] Y. Jiang and J. Ma. Combination features and models for human detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[61] J. L. Jones. Robots at the tipping point: the road to irobot roomba. *Robotics & Automation Magazine, IEEE*, 13(1):76–78, 2006.

[62] M. J. Jones and T. Poggio. Multidimensional morphable models. In *Computer Vision, 1998. Sixth International Conference on*, pages 683–688. IEEE, 1998.

[63] I. Kamon, E. Rivlin, and E. Rimon. A new range-sensor based globally convergent navigation algorithm for mobile robots. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 1, pages 429–435. IEEE, 1996.

[64] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006.

[65] M. Khatib and R. Chatila. An extended potential field approach for mobile robot sensor-based motions. In *Proc. International Conference on Intelligent Autonomous Systems (IAS4)*, 1995.

[66] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.

[67] G. Kim and W. Chung. Tripodal schematic control architecture for integration of multi-functional indoor service robots. *Industrial Electronics, IEEE Transactions on*, 53(5):1723–1736, 2006.

[68] K. Konolige, K. Myers, E. Ruspini, and A. Saffiotti. The saphira architecture: A design for autonomy. *Journal of experimental & theoretical artificial intelligence*, 9(2-3):215–235, 1997.

[69] M. Kothari, I. Postlethwaite, and D.-W. Gu. A suboptimal path planning algorithm using rapidly-exploring random trees. *International Journal of Aerospace Innovations*, 2(1):93–104, 2010.

[70] C.-H. Ku and W.-H. Tsai. Obstacle avoidance for autonomous land vehicle navigation in indoor environments by quadratic classifier. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 29(3):416–426, Jun 1999.

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

[72] B. Leibe, N. Cornelis, K. Cornelis, and L. Van Gool. Dynamic 3d scene analysis from a moving vehicle. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.

[73] D. Levi, S. Silberstein, and A. Bar-Hillel. Fast multiple-part based object detection using kd-ferns. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[74] K. Li, Q. Zhang, S. Kwong, M. Li, and R. Wang. Stable matching-based selection in evolutionary multiobjective optimization. *Evolutionary Computation, IEEE Transactions on*, 18(6):909–923, Dec 2014.

[75] L. Li, S. Yan, X. Yu, Y. K. Tan, and H. Li. Robust multiperson detection and tracking for mobile service and social robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(5):1398–1412, Oct 2012.

[76] Z. Lin and L. S. Davis. A pose-invariant descriptor for human detection and segmentation. In *Computer Vision–ECCV 2008*, pages 423–436. Springer, 2008.

[77] M. Lindström and J.-O. Eklundh. Detecting and tracking moving objects from a mobile platform using a laser range scanner. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 3, pages 1364–1369. IEEE, 2001.

[78] M. Lindström, A. Orebäck, and H. I. Christensen. Berra: A research architecture for service robots. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 4, pages 3278–3283. IEEE, 2000.

[79] J. Liu and J. Wu. *Multiagent Robotic Systems*. CRC press, 2001.

[80] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4(4):333–349, 1997.

[81] B. D. Lucas, T. Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981.

[82] V. Lumelsky and T. Skewis. Incorporating range sensing in the robot navigation function. *Systems, Man and Cybernetics, IEEE Transactions on*, 20(5):1058–1069, 1990.

[83] V. J. Lumelsky and A. A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2(1-4):403–430, 1987.

[84] P. Luo, Y. Tian, X. Wang, and X. Tang. Switchable deep network for pedestrian detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[85] R. C. Luo, Y. J. Chen, C. T. Liao, and A. C. Tsai. Mobile robot based human detection and tracking using range and intensity data fusion. In *Advanced Robotics and Its Social Impacts, 2007. ARSO 2007. IEEE Workshop on*, pages 1–6. IEEE, 2007.

[86] M. Maimone, Y. Cheng, and L. Matthies. Two years of visual odometry on the mars exploration rovers. *Journal of Field Robotics*, 24(3):169–186, 2007.

[87] J. Marin, D. Vázquez, A. Lopez, J. Amores, and B. Leibe. Random forests of local experts for pedestrian detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2592–2599, 2013.

[88] S. Munder, C. Schnorr, and D. M. Gavrila. Pedestrian detection and tracking using a mixture of view-based shape–texture models. *IEEE Transactions on Intelligent Transportation Systems*, 9(2):333–343, 2008.

[89] L. Muratet, S. Doncieux, Y. Briere, and J.-A. Meyer. A contribution to vision-based autonomous helicopter flight in urban environments. *Robotics and Autonomous Systems*, 50(4):195–209, 2005.

[90] C. Nakajima, M. Pontil, B. Heisele, and T. Poggio. Full-body person recognition system. *Pattern recognition*, 36(9):1997–2006, 2003.

[91] D. R. Nelson, D. B. Barber, T. W. McLain, and R. W. Beard. Vector field path following for miniature air vehicles. *Robotics, IEEE Transactions on*, 23(3):519–529, 2007.

[92] I. A. Nesnas, A. Wright, M. Bajracharya, R. Simmons, and T. Estlin. Claraty and challenges of developing interoperable robotic software. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 3, pages 2428–2435, 2003.

[93] M. Neves and E. Oliveira. A multi-agent approach for a mobile robot control system. In *Proceedings of Workshop on Multi-Agent Systems: Theory and Applications(MASTA97-EPPIA97)-Coimbra-Portugal*, pages 1–14, 1997.

[94] D. Nitzan, A. E. Brain, and R. O. Duda. The measurement and use of registered reflectance and range data in scene analysis. *Proceedings of the IEEE*, 65(2):206–220, 1977.

[95] T. Noonan, J. Fisher, and B. Bryant. Autonomous lawn mower, Apr. 20 1993. US Patent 5,204,814.

[96] H. S. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11:205–244, 1996.

[97] P. O'Donovan. Optical flow: Techniques and applications. *The University of Saskatchewan, TR*, 502425, 2005.

[98] W. Ouyang and X. Wang. A discriminative deep model for pedestrian detection with occlusion handling. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 3258–3265, 2012.

[99] W. Ouyang and X. Wang. Joint deep learning for pedestrian detection. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2013.

[100] W. Ouyang and X. Wang. Single-pedestrian detection aided by multi-pedestrian detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[101] C. Papageorgiou and T. Poggio. A trainable system for object detection. *International Journal of Computer Vision*, 38(1):15–33, 2000.

[102] S. Park, J. Deyst, and J. P. How. A new nonlinear guidance logic for trajectory tracking. In *AIAA guidance, navigation, and control conference and exhibit*, pages 16–19, 2004.

[103] S. Park, J. Deyst, and J. P. How. Performance and lyapunov stability of a nonlinear path following guidance method. *Journal of Guidance, Control, and Dynamics*, 30(6):1718–1728, 2007.

[104] L. E. Parker. Alliance: An architecture for fault tolerant multirobot cooperation. *Robotics and Automation, IEEE Transactions on*, 14(2):220–240, 1998.

[105] R. Polana and R. Nelson. Low level recognition of human motion (or how to get your man without finding his body parts). In *Motion of Non-Rigid and Articulated Objects, 1994., Proceedings of the 1994 IEEE Workshop on*, pages 77–82. IEEE, 1994.

[106] M. Quigley, E. Berger, A. Y. Ng, et al. Stair: Hardware and software architecture. In *AAAI 2007 Robotics Workshop, Vancouver, BC*, pages 31–37, 2007.

[107] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[108] I. Rhee, S. Park, and C.-K. Ryoo. A tight path following algorithm of an uas based on pid control. In *SICE Annual Conference 2010, Proceedings of*, pages 1270–1273. IEEE, 2010.

[109] M. Rioux, F. Blais, J. Beraldin, and P. Boulanger. Range imaging sensors development at nrc laboratories. In *Interpretation of 3D Scenes, 1989. Proceedings., Workshop on*, pages 154–160. IEEE, 1989.

[110] S. Rockel, D. Klimentjew, and J. Zhang. A multi-robot platform for mobile robotsa novel evaluation and development approach with multi-agent technology. In *Multisensor Fusion and Integration for Intelligent Systems (MFI), 2012 IEEE Conference on*, pages 470–477, 2012.

[111] J. K. Rosenblatt. Damn: A distributed architecture for mobile navigation. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2-3):339–360, 1997.

[112] S. Russell, P. Norvig, and A. Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27, 1995.

[113] P. Sabzmeydani and G. Mori. Detecting pedestrians by learning shapelet features. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[114] N. Sariff and N. Buniyamin. An overview of autonomous mobile robot path planning algorithms. In *2006 4th Student Conference on Research and Development*, pages 183–188. IEEE, 2006.

[115] D. Schulz, W. Burgard, D. Fox, and A. B. Cremers. People tracking with mobile robots using sample-based joint probabilistic data association filters. *The International Journal of Robotics Research*, 22(2):99–116, 2003.

[116] J. T. Schwartz and M. Sharir. On the piano movers' problem i. the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on pure and applied mathematics*, 36(3):345–398, 1983.

[117] E. Seemann, M. Fritz, and B. Schiele. Towards robust pedestrian detection in crowded image sequences. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.

[118] T. Serre, L. Wolf, and T. Poggio. Object recognition with features inspired by visual cortex. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 994–1000. IEEE, 2005.

[119] A. Shashua, Y. Gdalyahu, and G. Hayun. Pedestrian detection for driving assistance systems: Single-frame classification and system level performance. In *Intelligent Vehicles Symposium, 2004 IEEE*, pages 1–6. IEEE, 2004.

[120] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[121] R. G. Simmons. Structured control for autonomous robots. *Robotics and Automation, IEEE Transactions on*, 10(1):34–43, 1994.

[122] S. M. Smirnakis, M. J. Berry, D. K. Warland, W. Bialek, M. Meister, et al. Adaptation of retinal processing to image contrast and spatial scale. *Nature*, 386(6620):69–73, 1997.

[123] M. Spengler and B. Schiele. Towards robust multi-cue integration for visual tracking. *Machine Vision and Applications*, 14(1):50–58, 2003.

[124] B. Stenger, A. Thayananthan, P. H. Torr, and R. Cipolla. Model-based hand tracking using a hierarchical bayesian filter. *IEEE transactions on pattern analysis and machine intelligence*, 28(9):1372–1384, 2006.

[125] P. Sujit, S. Saripalli, and J. Borges Sousa. Unmanned aerial vehicle path following: A survey and analysis of algorithms for fixed-wing unmanned aerial vehicless. *Control Systems, IEEE*, 34(1):42–59, 2014.

[126] S. Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.

[127] K. P. Sycara. Multiagent systems. *AI magazine*, 19(2):79, 1998.

[128] L. F. Tammero and M. H. Dickinson. The influence of visual landscape on the free flight behavior of the fruit fly drosophila melanogaster. *Journal of Experimental Biology*, 205(3):327–343, 2002.

[129] J.-Y. Tigli and M. Thomas. Use of multi agent systems for mobile robotics control. In *Systems, Man, and Cybernetics, 1994. Humans, Information*

and Technology., 1994 IEEE International Conference on, volume 1, pages 588–592, 1994.

[130] M. Tistarelli, E. Grosso, and G. Sandini. Dynamic stereo in visual navigation. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pages 186–193, 1991.

[131] K. Toyama and A. Blake. Probabilistic tracking with exemplars in a metric space. *International Journal of Computer Vision*, 48(1):9–19, 2002.

[132] D. Tran and D. A. Forsyth. Configuration estimates improve pedestrian finding. In *Advances in neural information processing systems*, pages 1529–1536, 2008.

[133] A. Treptow, G. Cielniak, and T. Duckett. Real-time people tracking for mobile robots using thermal vision. *Robotics and Autonomous Systems*, 54(9):729–739, 2006.

[134] I. Ulrich and J. Borenstein. Vfhˆ*: Local obstacle avoidance with look-ahead verification. In *ICRA*, pages 2505–2511, 2000.

[135] N. Vandapel, J. Kuffner, and O. Amidi. Planning 3-d path networks in unstructured environments. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 4624–4629. IEEE, 2005.

[136] P. Viola, M. J. Jones, and D. Snow. Detecting pedestrians using patterns of motion and appearance. *International Journal of Computer Vision*, 63(2):153–161, 2005.

[137] B. Wang, X. Dong, and B. M. Chen. Cascaded control of 3d path following for an unmanned helicopter. In *Cybernetics and Intelligent Systems (CIS), 2010 IEEE Conference on*, pages 70–75. IEEE, 2010.

[138] X. Wang, T. X. Han, and S. Yan. An hog-lbp human detector with partial occlusion handling. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 32–39. IEEE, 2009.

[139] R. Wilcox, S. Nikolaidis, and J. Shah. Optimization of temporal dynamics for adaptive human-robot interaction in assembly manufacturing. *Robotics*, page 441, 2013.

[140] A. Willms and S. X. Yang. An efficient dynamic system for real-time robot-path planning. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 36(4):755–766, Aug 2006.

[141] J. D. Woll. Vorad collision warning radar. In *Radar conference, 1995., Record of the IEEE 1995 International*, pages 369–372. IEEE, 1995.

[142] M. Wooldridge, N. R. Jennings, et al. Intelligent agents: Theory and practice. *Knowledge engineering review*, 10(2):115–152, 1995.

[143] B. Wu and R. Nevatia. Detection and tracking of multiple, partially occluded humans by bayesian combination of edgelet based part detectors. *International Journal of Computer Vision*, 75(2):247–266, 2007.

[144] J. Wu, C. Geyer, and J. M. Rehg. Real-time human detection using contour cues. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 860–867. IEEE, 2011.

[145] P. R. Wurman, R. D'Andrea, and M. Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 29(1):9, 2008.

[146] B. M. Yamauchi. Packbot: a versatile platform for military robotics. In *Defense and Security*, pages 228–237. International Society for Optics and Photonics, 2004.

[147] J.-M. Yang and J.-H. Kim. Sliding mode control for trajectory tracking of nonholonomic wheeled mobile robots. *IEEE Transactions on Robotics and Automation*, 15(3):578–587, 1999.

[148] C. Yap. Algorithmic and geometric aspects of robotics, chapter algorithmic motion planning. jt schwartz and ck yap, 1987.

[149] H. Yavuz and A. Bradshaw. A new conceptual approach to the design of hybrid control architecture for autonomous mobile robots. *Journal of Intelligent and Robotic Systems*, 34(1):1–26, 2002.

[150] J. S. Zelek. Dynamic path planning. In *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on*, volume 2, pages 1285–1290. IEEE, 1995.

[151] X. Zeng, W. Ouyang, and X. Wang. Multi-stage contextual deep learning for pedestrian detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 121–128, 2013.

[152] W. Zhang, G. Zelinsky, and D. Samara. Real-time accurate object detection using multiple resolutions. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[153] T. Zhao and R. Nevatia. Tracking multiple humans in complex situations. *IEEE transactions on pattern analysis and machine intelligence*, 26(9):1208–1221, 2004.

[154] S. Zingg, D. Scaramuzza, S. Weiss, and R. Siegwart. Mav navigation through indoor corridors using optical flow. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3361–3368, 2010.