UNIVERSITÀ DEGLI STUDI DI PARMA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Dottorato di Ricerca in Tecnologie dell'Informazione

XX Ciclo

Stefano Cattani

STUDIO DI UN'ARCHITETTURA SOFTWARE PER VEICOLI AUTONOMI BASATI SU SENSORI DI VISIONE

DISSERTAZIONE PRESENTATA PER IL CONSEGUIMENTO

DEL TITOLO DI DOTTORE DI RICERCA

GENNAIO 2008

Indice

Introduzione

La ricerca nel campo dei veicoli autonomi di superficie di tipo convenzionale, basati cioè su automobili o camion per uso civile, ha vissuto un importante ritorno di interesse negli ultimi anni, soprattutto grazie alle iniziative del Defence Advanced Research Projects Agency (DARPA) del governo americano, chiamate *Grand Challenge* [1] [2] [3]. La prima edizione di questa manifestazione, svoltasi lungo le strade desertiche tra California e Nevada, ha visto la partecipazione preponderante di istituzioni universitarie, in particolare quelle che storicamente vantavano una certa esperienza nel campo dello sviluppo di veicoli robotici, come Carnagie Mellon, Calthec, Ohio State University. In quel contesto la presenza di rappresentati dell'industria privata era minima, e quasi esclusivamente relegata al ruolo di finanziatori. A quel tempo, che oggi sembra più lontano di quanto non sia in realtà, questo tipo di operazioni erano ancora avvolte da un'aura di "bizzarria universitaria", i veicoli apparivano, e spesso erano nella sostanza, estremamente artigianali ed approssimativi, quasi sempre realizzati da laureandi in pochi mesi di lavoro e studio, e la copertura mediatica era minima e caratterizzata da un certo sarcasmo misto a scetticismo.

Osservando l'elenco dei partecipanti all'ultima edizione, chiamata Urban Challenge e svoltasi nel 2007 in un'ex base militare della California, si può osservare come questo scenario sia cambiato radicalmente: il tema della dello sviluppo di sistemi in grado di automatizzare la guida di un veicolo in ambiente misto urbano ed extra urbano raccoglie ormai l'interesse delle maggiori case automobilistiche mondiali, le quali partecipano direttamente con i loro veicoli, i loro gruppi di sviluppo e,

non ultimi, un notevole impegno di marketing e pubbliche relazioni. La ragione di questo repentino cambiamento è semplice: è da questo settore della ricerca che, con ogni probabilità, nasceranno le idee, i prototipi, nonché i migliori gruppi di ricerca, che verranno impiegati sui veicoli civili delle prossime generazioni, per lo sviluppo dei ben noti Advanced Driver Assistance Systems (ADAS). Assieme allo sviluppo di motori in grado di sfruttare combustibili alternativi, i sistema di ausilio alla guida rappresentano, attualmente, la più promettente opportunità di evoluzione dei veicoli commerciali.

La mia attività di Dottorato si è svolta prevalentemente nell'ambito delle varie Grand Challenge che si sono succedute nel corso di questi anni, come membro del laboratorio di visione artificiale VisLab dell'Università di Parma. Il nostro gruppo si è occupato dello sviluppo di sistemi di visione artificiale in grado di percepire il mondo circostante il veicolo, rilevando elementi come ostacoli [4] [5], confini della carreggiata, linee di demarcazione delle corsie, linee di stop. Parallelamente allo sviluppo di sistemi di visione, nel corso di questi anni mi sono occupato anche dell'integrazione tra i sistemi di visione artificiale sviluppati a Parma e il software di guida automatica presente sul veicolo.

Durante quest'ultima attività ho avuto l'opportunutà di osservare le diverse soluzioni architetturali adottate per integrare i componenti che formano l'intero sistema veicolo (pianificazione del moto+sensori+sistema di navigazione), la loro evoluzione nel corso delle varie edizioni delle Grand Challenge, i loro pregi, i loro difetti. Ho potuto notare come l'influenza di tali scelte sulle prestazioni finali del veicolo autonomo sia notevole: lo sviluppo di questo tipo di sistemi è spesso difficoltoso e convulso, sia per la complessità intrinseca del problema, sia per le difficoltà di pianificazione di un lavoro che rimane, in tutto e per tutto, un'attività di ricerca pura. Inoltre le competenze richieste per la realizzazione di sistemi così complessi sono varie e numerose, includendo l'intelligenza artificiale, l'elettronica, la meccanica, le telecomunicazioni; questo porta inevitabilmente alla formazione di team di sviluppo molto eterogenei, dove si possono facilmente mescolare esperienze professionali e forma-

tive completamente diverse. La definizione stessa dei confini tra un sottosistema e l'altro è spesso incerta e parziale: un sensore può essere chiamato a rilevare elementi del mondo che inizialmente non erano previsti dalle sue specifiche, o che sembravano essere di secondaria importanza; alcuni sottosistemi posso essere completamente eliminati dal sistema perchè le loro prestazioni vengono ritenute non sufficienti; nuovi componenti possono subentrare in qualunque fase dello sviluppo o dei test. È chiaro come, in questo contesto, la definizione di un'architettura software adeguata a gestire la complessità, l'eterogeneità e la fluidità del sistema sia fondamentale per il successo del team.

Sulla base di queste esperienze e considerazioni, ho sviluppato una mia personale proposta di architettura software per veicoli autonomi, che costituisce il tema principale di questa tesi di Dottorato. È bene sottolineare che non si tratta di un framework di sviluppo per veicoli autonomi, il quale è normalmente definito in termini di Application Program Interface (API), tale da rappresentare uno strumento operativo per la realizzazione di questi sistemi. Quello che si tenta di fare in questo lavoro è di fornire una *descrizione esterna* di quelli che dovrebbero essere i componenti principali di un veicolo autonomo, almeno dal punto di vista software, attraverso la descrizione del modo in cui questi *comunicano* e si *coordinano* reciprocamente. Inoltre, viene presentato un possibile insieme di questi componenti, descrivendone le principali caratteristiche ed i principali compiti.

Per concludere, nel Capitolo 4 sono presentati alcuni sistemi di Visione Artificiale di cui mi sono occupato direttamente durante il corso di Dottorato, e che sono stati installati e testati su veicoli alcuni autonomi.

Capitolo 1

Problematiche dei veicoli autonomi

In questo Capitolo verrà illustrato cosa si intende per veicolo autonomo, e quali sono le problematiche principali nello sviluppo di questi sistemi. Si vedranno inoltre alcuni esempi di celebri veicoli autonomi della storia.

1.1 Cos'è un veicolo autonomo

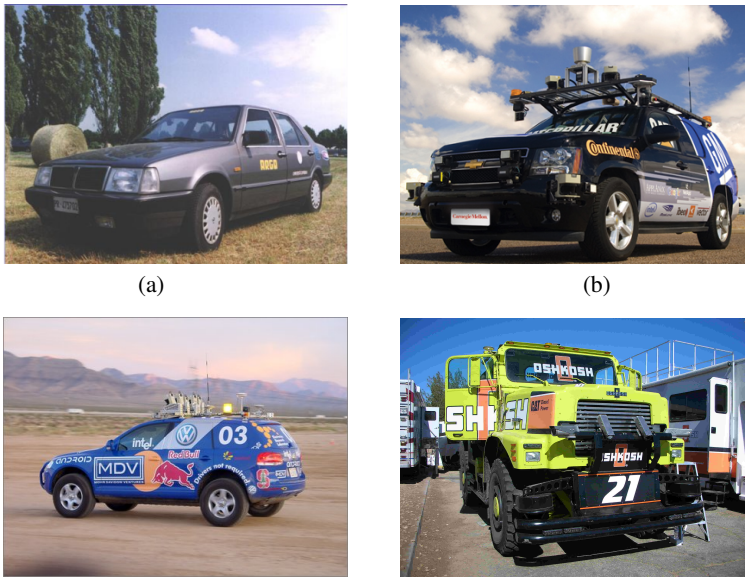
Per poter comprendere a fondo quali siano le caratteristiche che dovrebbe avere una buona architettura software a loro dedicata, dobbiamo per prima cosa definire con chiarezza cosa si intenda per "veicoli autonomi":

> Si definisce **Veicolo Autonomo** un mezzo in grado di muoversi in un ambiente *dinamico* e *sconosciuto*, allo scopo di raggiungere un punto di arrivo passando attraverso alcuni passaggi obbligati (*waypoints*), il tutto senza l'intervento di operatori umani, ne' diretto ne' remoto. Il movimento deve avvenire senza provocare indicenti e nella massima sicurezza possibile.

La definizione è abbastanza generale, ed include numerosi tipologie di veicoli. Pensiamo agli aerei ad esempio, che già vantano un elevato livello di automazione nel loro controllo, alle navi, ai sottomarini, specialmente quelli per lo studio degli abissi, e non ultimi i veicoli per l'esplorazione planetaria. Questo tipo di veicoli è tipicamente chiamato a muoversi in ambienti molto ben definiti, dove le interazioni con l'esterno sono limitate, ed in un certo senso prevedibili, facilitando il compito dei ricercatori. Diverso è il discorso che riguarda lo sviluppo di veicoli autonomi *comuni*, come la automobili per il trasporto dei passeggeri, o i mezzi per il trasporto su strada delle merci, come i camion. Questa tipologia di mezzi deve normalmente interagire con un ambiente estremamente complesso, dinamico e del tutto imprevedibile, nel quale, alle volte, anche gli esseri umani incontrano delle difficoltà. Inoltre, la presenza di persone nelle vicinanze dei mezzi durante la guida automatica rende particolarmente delicata la questione sicurezza. Lo scenario è radicalmente cambiato con l'arrivo delle DARPA Grand Challenge che, nate sotto la spinta di esigenze prevalentemente militari, hanno in qualche modo aperto gli occhi alla comunità scientifica, mostrando che non è così impossibile la realizzazione di veicoli in grado di muoversi in ambienti urbani. Certo, lo stato dell'arte attuale mostra veicoli che troverebbero ancora delle difficoltà ad integrarsi nel traffico cittadino, ma sono stati comunque fatti passi da gingante nella giusta direzione.

In Figura 1.1 sono mostrati alcuni celebri veicoli autonomi di questo tipo.

Quella illustrata in questa tesi è un'architettura rivolta proprio a quest'ultima tipologia di veicoli, quelli progettati per muoversi in ambienti urbani ed extra urbani, sulle comuni strade presenti sul territorio, con il compito di interagire correttamente con altri veicoli, e di rispettare il codice della strada.



(a) (b)

(c) (d)

Figura 1.1: Alcuni veicoli autonomi di superficie derivati da mezzi commerciali: (a) Argo, sviluppato dall'Università di Parma; (b) Boss, sviluppato dalla Carnagie Mellon University e vincitore della Urban Challenge 2007; (c) Stanley, sviluppato dalla Stanford University e vincitore della Grand Challenge 2005; (d) TerraMax, sviluppato dalla Oshkosh Truck in collaborazione con l'Università di Parma, e finalista alla Urban Challenge 2007.

1.2 Sviluppare un veicolo autonomo

Nel corso di questi anni ho partecipato allo sviluppo di diversi veicoli autonomi, avendo modo di osservare dall'interno quali sono le principali difficoltà che si incontrano in questo tipo di attività. Per quanto possa apparire una considerazione banale, il problema principale da affrontare è la *complessità* generale del progetto, che si manifesta principalmente nei seguenti modi:

- *molte competenze necessarie*: per realizzare un veicolo autonomo sono necessarie competenze che spaziano dalla meccanica, all'elettronica, all'intelligenza artificiale fino alla visione artificiale. Inoltre è necessario avere esperienza nell'uso dei numerosi sensori di cui è dotato, come telecamere, laser, antenne GPS.

- *gruppo di sviluppo eterogeneo*: conseguenza diretta del punto precedente è la formazione di gruppi di sviluppo molto eterogenei, i quali probabilmente adotteranno linguaggi di programmazione diversi, piattaforme hardware differenti, diversi sistemi operativi. Essi andranno integrati tra loro nel modo più semplice ed efficiente possibile.

- *architettura fluida*: il carattere fortemente di ricerca di questo tipo di attività fa si che il sistema tenda a cambiare di frequente, con il progressivo prendere coscienza, da parte del team, delle reali funzionalità richieste. Non è inusuale che, dopo mesi di sviluppo, determinate parti del sistema vengano completamente riprogettate, o sostituite. L'architettura deve essere in grado gestire questo tipo di situazioni in modo da minimizzare le perdite di tempo

- *difficoltà di debug*: il debug di un veicolo autonomo è particolarmente critico, in quanto esso è generalmente un "pezzo unico", una singola risorsa alla quale tutti i componenti del gruppo necessitano di accedere. Raramente si ha a disposizione un secondo veicolo, clone del primo, su cui effettuare test in parallelo. Emerge quindi la necessità di avere a disposizione qualche meccanismo per il

debug *off-line*, attraverso la possibilità di *simulare*, tutto o in parte, il sistema veicolo.

Per gestire la complessità del sistema, nei termini illustrati dalla considerazioni precedenti, si è creduto utile dividere il sistema software in *moduli* il più possibile indipendenti. Questi moduli dovranno essere facilmente sviluppabili in ambienti differenti, facilmente integrabili tra loro e intercambiabili. Inoltre dovrà essere possibile simulare la presenza di uno qualunque dei moduli, in modo da poter effettuare delle sessioni di debug anche in laboratorio.

Vedremo nei prossimi Capitoli come queste linee guida si tradurranno negli elementi costituivi dell'architettura proposta.

1.3 Elementi software di un veicolo autonomo

Chiudiamo questo capitolo illustrando quali sono gli elementi software che più di frequente sono presenti su un veicolo autonomo. In Figura 1.2 vediamo una schematizzazione di tali elementi, in una tipica struttura a livelli, dove ogni strato dipende da quelli che si trovano sotto, ma non da quelli che si trovano sopra.

Alla sommità della struttura troviamo i componenti di alto livello che si occupano di creare un'interfaccia verso l'utente, di gestire l'avvio e l'arresto della missione e di sovrintendere all'elaborazione dei moduli sottostanti. Il livello chiamato "Autonomous Behaviour" comprende tutti quegli elementi software che realizzano la guida vera e propria del veicolo, come il pianificatore del moto, la fusione sensoriale, il tracking degli ostacoli percepiti, ecc.. Essi comunicheranno all'interfaccia utente informazioni sullo stato corrente della missione dal punto di vista della guida automatica, come la traiettoria desiderata, o l'attuale modalità di guida adottata (ad esempio quella di aggiramento ostacolo, o di parcheggio). Al disotto dell'"Autonomous Behaviour" troviamo i componenti software che elaborano i dati provenienti direttamente dai dispositivi hardware, fornendo per essi anche un'interfaccia di alto livello. Tra i dispositivi hardware includiamo anche il veicolo stesso, il quale, per essere guidato, richiede che almeno un componente software sia in grado di agire direttamente

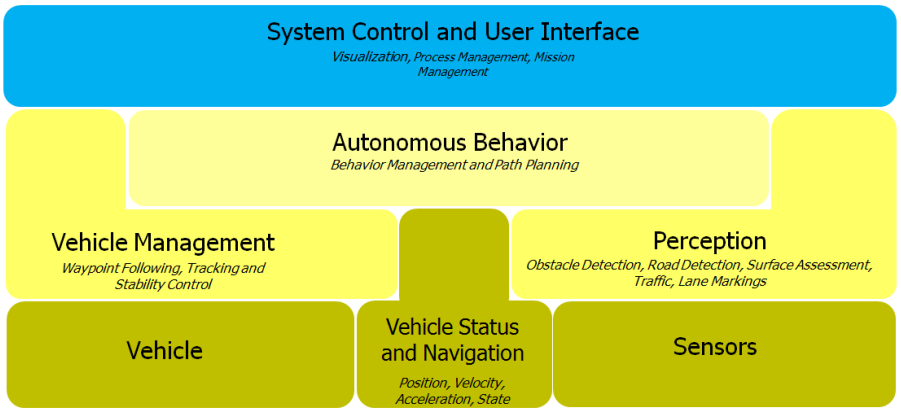


Figura 1.2: Schematizzazione degli elementi software tipici di un veicolo autonomo.

sui suoi attuatori, traducendo i comandi di alto livello ricevuti dal pianificatore nei corrispondenti comandi per il sistema "x-by-wire". A questo livello troviamo anche il software che si occupa di elaborare i dati provenienti dai sensori, come ad esempio gli algoritmi di visione artificiale. All'ultimo livello troviamo invece i dispositivi fisici. Notiamo come si sia voluto creare un collegamento diretto tra il sistema di navigazione e l'"Autonomous Behaviour": questo ha principalmente lo scopo di creare una differenziazione tra i servizi che fungono da driver dei dipositivi di navigazione (es. GPS), e quelli che invece gestiscono ed elaborano i sensori di percezione. Questi ultimi implementano funzionalità piuttosto complesse di elaborazione dati, trasformando il dato grezzo proveniente dal dispositivi fisici in una serie di informazioni di alto livello, come può essere la posizione e la forma di un ostacolo. Al contrario, il servizio di navigazione tipicamente fornisce solamente un'interfaccia verso di dispositivi, esegue cioè una mera traduzione tra due protocollo differenti, senza nessun valore aggiunto. Ecco perchè è sembrato più corretto assimilare questo servizio al livello fisico.

Come si può notare la descrizione è del tutto generale, non propone nulla di realmente innovativo, ma descrive con chiarezza quali siano le marco aree in cui possiamo suddividere il software necessario ad un veicolo autonomo. Sulla base di

questa schematizzazione vedremo nei prossimi Capitoli una proposta di moduli che andranno a implementarlo.

Capitolo 2

Un proposta di architettura software

In questo capitolo inizia la descrizione nel dettaglio dell'architettura per veicoli autonomi proposta. Verranno in particolare illustrati i principali servizi di cui è composta, e i relativi schemi di controllo e comunicazione.

2.1 Struttura a servizi

Nel Capitolo precedente abbiamo illustrato quali sono i componenti principali che costituiscono il sistema software di un tipico veicoli autonomo. Abbiamo anche individuato quali sono le caratteristiche che un'architettura software per questo tipo dovrebbe avere, tra le quali la più importante è certamente la separazione tra descrizione esterna e implementazione specifica. Questo concetto ha guidato l'intero sviluppo della piattaforma proposta, portando alla definizione del suo elemento di base in funzione unicamente della sua descrizione esterna, che d'ora in avanti definiremo *interfaccia*. In particolare sono previste due tipi di interfacce che ogni componente deve necessariamente implementare:

- interfaccia di *controllo*: quella che definisce le interazione tra processi, in par-

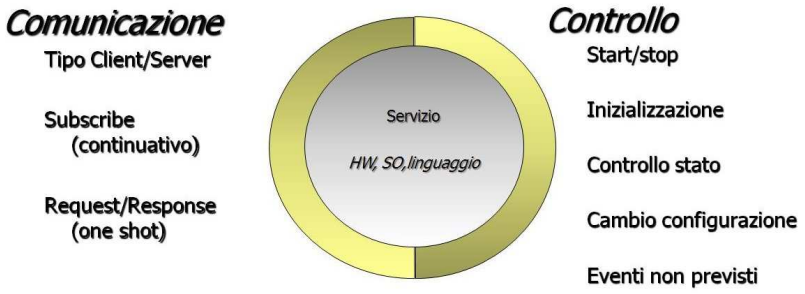


Figura 2.1: Schematizzazione della struttura ad interfacce di un servizio.

ticolare per quanto riguarda l'inizializzazione, l'avvio, l'arresto, la riconfigurazione, la gestione degli eventi aperiodici. La supervisione di queste operazioni viene effettuata attraverso un gestore dei processi/componenti di alto livello.

- interfaccia di *comunicazione*: quella che definisce le modalità con le quali i servizi scambiano dati utili per l'elaborazione specifica di ogni processo, in un'ottica di tipo produttore/consumatore.

Ognuna di queste interfacce, che nei Paragrafi del prossimo Capitolo 3.5 e 3.4 descriveremo nel dettaglio sia nella sintassi che nelle sequenze operative, definisce il modo in cui ogni processo che implementa un particolare componente del veicolo autonomo debba "relazionarsi" con l'esterno. Con questo termine intendiamo due aspetti in particolare: le modalità con cui avviene la richiesta e fornitura di informazioni prodotte dal componente stesso (immaginiamo ad esempio un sensore), ma anche come avviene il coordinamento, il controllo e la diagnostica che il supervisore della missione deve attuare sull'insieme dei componenti in esecuzione.

Ma veniamo ora a dare una definizione formale dell'elemento base che compone la nostra architettura, che d'ora in avanti chiameremo *servizio*:

Servizio è un qualunque processo, od insieme di processi, che implementa le interfacce di *controllo* e *comunicazione* definite dalle specifiche dell'architettura. Ogni servizio è univocamente identificato dai seguenti attributi:

- Indirizzo IP e porta sui quali è raggiungibile;
- Nome e descrizione;
- Elenco degli altri servizi del sistema da questo cui dipende.

Service Oriented Architecture e Sistemi Distribuiti. Vale la pena di sgombrare immediatamente il campo da equivoci sul significato e le implicazioni del termine *servizio*: esso non vuole necessariamente riferirsi, o richiamare, le così dette Service Oriented Architecture (SOA) [6], o il Service Oriented Computing (SOC) [7]. Questo tipo di architetture, largamente impiegate nell'ambito dei servizi Web, sono state piuttosto una fonte di ispirazione nella fase iniziale dello sviluppo, in particolare per quanto riguarda l'idea di vedere il complesso sistema di applicazioni, necessarie per l'automatizzazione di un veicolo, attraverso un insieme di servizi raggiungibili e coordinabili via rete. Successivamente, le peculiarità del particolare ambito per cui viene proposta questa architettura hanno portato alla modifica di questa impostazione, almeno per alcuni aspetti. Ricordiamo, ad esempio, la necessità di introdurre meccanismi di sicurezza di basso livello che siano indipendenti dalla normale interfaccia di controllo, tramite i quali sia possibile arrestare il veicolo in caso di emergenza. Questo è utile proprio in quei casi dove i normali sistemi di sicurezza, *comunque* implementati dai componenti di alto livello, falliscano. Immaginiamo, ad esempio, cosa avverrebbe se il servizio che si occupa di fungere da driver verso il veicolo dovesse restare isolato dal resto del sistema: in questo caso gli attuatori sarebbero totalmente inaccessibili dal software di alto livello, al quale non resterebbe nessuno strumento per agire sui comandi del veicolo, anche quando fosse in grado di rilevare la mancanza di comunicazione con il driver. Di conseguenza la nostra architettura dovrà prevedere qualche meccanismo di comunicazione, questa volta non più indipendente

dalla piattaforma hardware e dall'implementazione, con il quale il controllore di alto livello possa inviare dei comandi di emergenza al veicolo; questi meccanismi non sono rappresentabili in termini di architettura a servizi, e richiedono quindi una deroga a questo formalismo: il servizio di più alto livello deve avere accesso, tramite qualche collegamento dedicato, al servizio di interfaccia con il veicolo e potervi inviare i comandi necessari. Naturalmente, a fianco di questo meccanismo di sicurezza software, dovrà essere presente anche un circuito, realizzato in hardware, per l'arresto di emergenza del veicolo, da azionarsi manualmente da parte di un operatore umano. Quest'ultimo sistema non fa parte dell'architettura presentata, che si vuole collocare unicamente a livello software.

Nei prossimi due Paragrafi vedremo una carrellata dei principali servizi previsti nel sistema proposto. Quelli presentati in Paragrafo 2.2, come risulterà evidente, sono derivati direttamente dalla Figura 1.2, e caratterizzati dal fatto che tra di loro intercorrono relazioni basate principalmente sullo scambio di informazioni di elaborazione. Questo primo gruppo rappresenta gli elementi funzionali veri e propri, quelli che realizzano la guida automatica. Nel Paragrafo 2.3 vedremo invece i servizi che sovrintendono alla gestione dei servizi stessi, coordinandone le azioni, monitorandone la presenza e lo stato.

2.2 Schema di comunicazione

In questo Paragrafo andiamo ad illustrare una serie di servizi che interagiscono tra loro principalmente attraverso lo scambio delle informazioni che sono in grado di produrre. Questo schema può anche essere definito *funzionale*, in quanto mette in evidenza come un servizio, per eseguire le proprie funzioni, dipenda quasi sempre dai dati che gli vengono inviati da un altro. I dettagli implementativi sul modo in cui avviene lo scambio di dati è descritto dettagliatamente nel Capitolo 3; per ora limitiamoci a dare una descrizione dei servizi stessi e delle loro funzioni principali.

Osservando la Figura 2.2 vediamo che i servizi proposti sono i seguenti: *Au-*

tonomous Vehicle Manager, *World Perseption Server*, *Autonomous Vehicle Driver*, *Navigation System*, *Sensori*, *User Interface* e *Playback Server*.

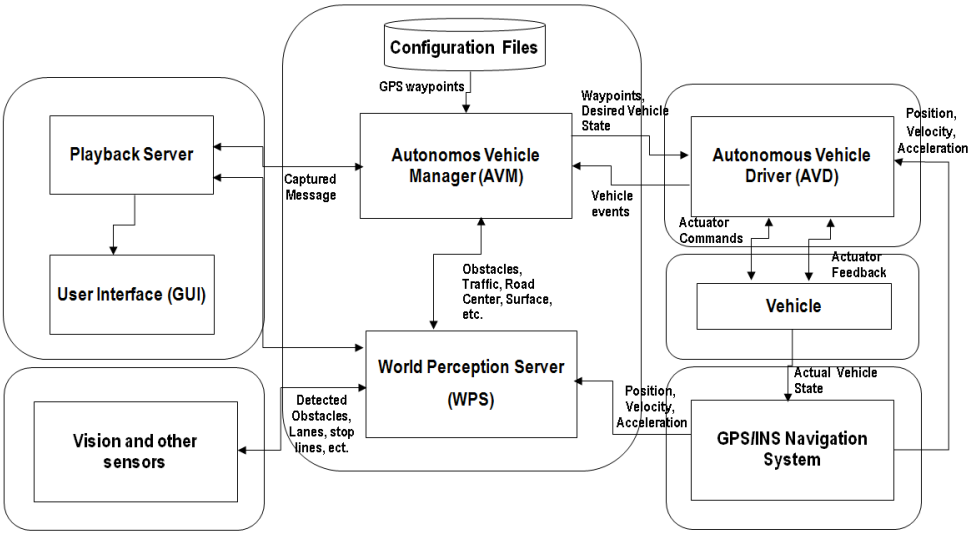


Figura 2.2: Esempio della struttura di comunicazione, con i relativi servizi coinvolti.

2.2.1 Autonomous Vehicle Manager

Questo è certamente il servizio fondamentale al livello attuale di analisi. L'Autonomous Vehicle Manager ha il compito di "guidare", nel senso più completo del termine, il veicolo:

- *caricamento waypoint e pre-pianificazione del moto*: il servizio carica la lista dei waypoints relativi alla missione, assieme a tutte le informazioni che sono disponibili staticamente sul percorso. In questa fase potrebbe anche essere prevista una pre-pianificazione del moto: noti i punti attraverso i quali il veicolo dovrà passare, è possibile costruire a priori una traiettoria *ideale* che li unisca tutti, impostando contestualmente le velocità nei vari tratti. Al presentarsi di eventuali ostacoli durante la marcia, che impediscano di seguire la traiettorie

ideale, verranno effettuate delle ripianificazioni del modo che andranno a costituire delle *deviazioni* dal percorso ideale, che verrà immediatamente ripreso dopo la manovra di obstacle avoidance. Questo approccio permette, nella prima fase, di realizzare un'ottimizzazione globale del moto del veicolo, ed allo stesso tempo è possibile codificare semplicemente come behaviour specifici, e locali, le situazioni che hanno richiesto la ripianificazione.

- *percezione dell'ambiente circostante*: interfacciandosi con il World Perseption Server (Paragrafo 2.2.2) esso riceverà una mappa delle informazioni sensoriali, che poi utilizzerà per pianificare il moto.

- *pianificazione locale del moto*: in questa fase vengono calcolati i percorsi locali di deviazione dalla traiettoria ideale, necessari tipicamente in caso di ostacoli.

- *behaviour management*: il servizio si occupa inoltre di modificare la modalità di guida corrente del veicolo. Il Vehicle Manager avrà il compito di individuare e implementare tutte le modalità di guida necessarie per affrontare le missioni previste per il veicolo autonomo che andiamo a sviluppare. Se, ad esempio, siamo in ambiente urbano, dovremo prevedere una modalità "incrocio", nella quale in veicolo dovrà agire nel seguente modo: 1) individuare la presenza e l'esatta posizione della linea di stop, corrispondente al punto in cui il veicolo è obbligato a fermarsi; 2) rilevare la conformazione dell'incrocio, localizzando le strade di uscita; 3) individuare i veicoli a cui è necessario dare la precedenza, ed attendere che liberino l'incrocio; 4) tenere traccia dei veicoli sui quali abbiamo precedenza, e controllare che la rispettino; 5) liberare l'incrocio. Una modalità di guida meno articolata, e valida per ambienti urbani ed extra urbani è quella di "obstacle avoidance": 1) modificare la velocità del veicolo non appena individuato un ostacolo che occupa la traiettoria ideale, eventualmente fino ad arrestare il moto; 2) pianificare una traiettoria alternativa come deviazione da quella ideale; 3) eseguire la manovra. Sempre in ambiente urbano possiamo proporre anche le seguenti modalità: parcheggio, sorpasso, immissione nel traffico con precedenza, cambio di corsia, incrocio

veicolo. La definizione completa di tutti i possibili behaviour è compito alla specifica implementazione dell'architettura, noto l'ambiente in cui il veicolo dovrà muoversi.

- *invio dei comando di guida*: interfacciandosi con il Autonomous Vehicle Driver (Paragrafo 2.2.3), invierà i comandi di alto livello al veicolo, per attuare le manovre pianificate. I comandi di alto livello inviati al driver del veicolo tipicamente riguarderanno lo spostamento in un punto specifico del mondo, l'impostazione della velocità, impostazione di particolari modalità di marcia o di fermata, come per esempio l'inserimento della marcia di parcheggio.

Possiamo dunque descrivere i compiti dell'Autonomous Vehicle Manager come il controllo del veicolo quando esso si trova in modalità automatica.

2.2.2 World Perseption Server

Il servizio che si occupa di raccogliere le informazioni prodotte dai sensori, e di riorganizzarle nella forma di una rappresentazione organica a completa del mondo circostante il veicolo, è il World Perseption Server:

- *raccolta informazioni sensoriali*: il servizio si connette ai servizi che gestiscono i sensori e, tramite i messaggi e le procedure definite dall'interfaccia di comunicazione, richiede ad essi di pubblicare le informazioni che producono. Come vedremo esistono due modalità in cui un sistema può richiedere informazioni ad un servizio: la prima per ottenere una fornitura continuativa di dati, a flusso diciamo, che terninerà con la richiesta esplicita di interruzione da parte del richiedente; la seconda invece prevede come risposta un singolo dato (one-shot), che, intuitivamente, può essere immaginato come il risultato di una singola scansione del mondo, fatta nell'instante in cui arriva la richiesta.

- *fusione sensoriale*: questo è uno dei compiti più importanti e delicati per il veicolo autonomo. I dati forniti dai diversi sensori devono infatti essere uniti

assieme, per ottenere un mappa del mondo circostante complessiva: date la percezioni di uno stesso oggetto, nel mondo fisico, fatte da diversi sensori con diverse capacità sensoriali, esse dovranno dare luogo ad un unico oggetto nella mappa del mondo inviata al pianificatore del moto, che tenga conto di tutte le informazioni disponibili. Allo stesso modo, se un sensore dovesse rilevare un falso positivo, esso dovrà essere filtrato in questa fase, incrociando i risultati degli altri sensori nella stessa zona. Ricade sulla fusione sensoriale anche il non facile compito del *tracking*.

- *pubblicazione mappa del mondo*: allo stesso modo di un sensore, il World Perseption Server implementerà l'interfaccia di comunicazione, tramite la quale metterà a disposizione degli altri servizi la mappa del mondo, sia in modalità one-shot che continuativa.

La scelta di assegnare la fusione sensoriale ad un servizio diverso da quello del path planning è dettata dalla necessità di rendere modulare il sistema, dividendo le mansioni che possono essere eseguite in modo indipendente. Si tratta quindi di una separazione di tipo *logico*, perchè l'architettura a servizi lascia la possibilità allo sviluppatore di incorporare il Word Perseption Server nello stesso processo, o insieme di processi, che formano l'Autonomous Vehicle Manager, per unificare in un unico blocco tutto il sistema di alto livello. Come detto infatti, il servizio è definito unicamente da un indirizzo IP, una porta, un nome ed un elenco di dipendenze, tutti elementi che possono essere gestiti da un unico processo per numerosi servizi contemporaneamente.

2.2.3 Autonomous Vehicle Driver

Questo servizio ha lo scopo di fornire un'interfaccia di alto livello per il veicolo fisico, permettendo il disaccopiamento tra il sistema di pianificazione del moto e l'attuazione fisica dei comandi conseguenti alla pianificazione stessa, rendendo l'architettura più facilmente adattabile a veicoli e contesti differenti. Altro compito dell'Au-

tonomous Vehicle Driver è fornire informazioni di tipo diagnostico all'alto livello. Vediamo alcuni dettagli:

- *interfaccia verso gli attuatori*: il servizio deve conoscere la dinamica del veicolo, ed essere in grado di modificarne i parametri di marcia attraverso l'interfaccia *x-by-wire*. La funzione più importante è certamente quella del *waypoint following*: il servizio deve essere in grado di comandare il veicolo in modo da portarlo, dalla posizione corrente, in un punto del mondo qualunque attraverso alcuni altri punti intermedi obbligatori. La composizione di questi punti intermedi spetta, naturalmente, al pianificatore del moto.

- *diagnostica veicolo*: è necessario fornire all'alto livello anche un'interfaccia con cui possa controllare lo stato del veicolo, ed essere avvisato in caso di problemi meccanici. Questo tipo di fornitura dati viene effettuata dall'Autonomous Vehicle Driver.

Per entrambi i compiti il servizio implementa l'interfaccia di comunicazione.

2.2.4 Navigation System

Il Navigation System può essere considerato come l'interfaccia verso i sistemi di navigazione e posizionamento geografico del veicolo. Virtualmente ogni veicolo autonomo dovrebbe essere in grado di localizzarsi nel mondo in cui agisce, per esempio tramite un sistema GPS. Allo stesso modo dovrebbe esserei in grado di registrare i dati di assetto del veicolo, come le accelerazioni, sia traslazionali che angolari, o le velocità calcolate tramite il sistema di posizionamento (velcità differenziali).

Questo tipo di dati sono utili a molti componenti del sistema software, come l'Autonomous Vehicle Driver, l'Autonomous Vehicle Manager ma anche ai sensori, che possono attuare strategie particolari di rilevazione dell'ambiente in base all'assetto del veicolo (come fa la visione, ad esempio [5]). Tutte le informazioni verranno scambiante mediante l'interfaccia di comunicazione.

2.2.5 Sensori

La sensoristica è un elemento fondamentale per un veicolo autonomo che voglia muoversi in un ambiente dinamico e sconosciuto. Esistono molti tipi di dispositivo in grado di percepire l'ambiente circostante, ogni uno con le proprie peculiarità, il proprio dominio di funzionamento, ed i propri limiti. Spesso sensori diversi hanno capacità di rilevare gli stessi elementi del mondo fisico, ma in modi diversi e con diverse accuratezze, ed è proprio per questa ragione che è prevista una fase di fusione sensoriale, la quale però può funzionare solo se l'output di questi dispositivi viene reso omogeneo rispetto ad una rappresentazione comune. Ecco quindi che ha senso formalizzare i sensori, ed i loro driver, attraverso uno specifico tipo di servizio e di interfaccia di comunicazione. Tale interfaccia prevederà i seguenti tipi di dato:

- *ostacoli frontali*: qualunque oggetto che si trovi ad intersecare la traiettoria del veicolo, emergendo dal terreno abbastanza da costituire un pericolo, e tale da dover essere trattato in modo speciale dal pianificatore del moto. In questa categoria potremmo includere anche gli ostacoli negativi, cioè le disconnessioni del terreno che formano buche tali da dover anch'esse tenute in considerazione dall'alto livello.

- *segnaletica orizzontale*: il veicolo deve essere in grado di rilevare la presenta di segnaletica orizzontale, in particolare due tipi sono molto importanti: 1) linee di demarcazione delle corsie, che permettono al veicolo di restare entro i confini della strada anche in caso di waypoint poco accurati; 2) stop line, per consentire al veicolo di fermarsi esattamente al confine dell'intersezione affrontata, che corrisponde al punto di massima visibilità possibile.

- *confini del percorso in fuori strada*: il veicolo potrebbe essere chiamato a percorrere strade dove non è presente nessuna pavimentazione di asfalto o cemento, neé alcun tipo di segnaletica orizzontale. In questo caso la ricerca dei confini della carreggiata diventa un problema completamente indipendente dalla ri-

cerca di segnaletica, e spesso viene chiamato *path detection*. Nel Paragrafo 4.1 del prossimo Capitolo vengono illustrati alcuni di questi sistemi.

- *marciapiedi*: in assenza di segnaletica orizzontante, in taluni casi è possibile dedurre la conformazione della strada usando i marciapiedi come loro confine laterale. Inoltre piò essere necessario rilevare i marciapiedi anche quando eseguiamo una manovra di inversione ad U: in questo caso vogliamo riconosce i marciapiedi in quanto rappresentano un confine trasversale della strada, utile in particolare durante la fase di retromarcia, per non uscire dalla carreggiata.

- *veicoli che sopraggiungono ad un incrocio*: per questo caso particolare di ostacoli, il veicolo autonomo è soprattutto interessato a rilevarne la velocità. Infatti, quando dobbiamo inserirci nel traffico dando la precedenza, è necessario lasciare una certa distanza di sicurezza tra i veicoli che sopraggiungono. Per fare questo dobbiamo ottenere una stima della velocità del veicolo, per prevederne il tempo di arrivo sull'incrocio. Nel Paragrafo 4.2 del prossimo Capitolo viene illustrato uno di questi sistemi.

Nel Paragrafo 3.4 del prossimo Capitolo vedremo nel dettaglio la sintassi di questi messaggi, che include implicitamente la rappresentazione scelta per i vari elementi del mondo fisico. Ad esempio un ostacolo sarà rappresentato tramite un poligono, il confine della carreggiata tramite una spezzata, ecc.

Anche questi servizi, come i precedenti, comunicano scambiano dati attraverso l'interfaccia di comunicazione.

2.2.6 User Interface

Il sistema di guida automatica necessiterà di interfacciarci con un operatore umano, per l'attivazione della guida automatica, per il controllo del comportamento del veicolo, per la diagnostica sui servizi, per monitorare lo stato di avanzamento della missione. Per questo è previsto un servizio che fornisca un'interfaccia grafica all'utente, dove queste informazioni siano disponibili nel loro complesso. Si tratta quindi

di un servizio in cui convergono molte informazioni diverse, come la percezione del mondo, i waypoints, lo stato del veicolo, la posizione nel mondo, lo stato dei servizi, ecc.; questi dati saranno ottenuti interrogando i servizi interessati (sensori, gestore del veicolo, navigatore, gestore dei servizi) attraverso l'interfaccia di comunicazione.

2.2.7 Playback Server

Concludiamo questa carrellata di servizi con uno molto particolare, che non può essere considerato necessario ad un veicolo autonomo, ma del quale vogliamo sottolineare l'utilità in fase di sviluppo. Il Playback Server ha il compito di registrare tutte le informazioni sensoriali prodotte durante una missione, per poi dare la possibilità di rileggerle successivamente, allo scopo di simulare qualche servizio. La possibilità di simulare un servizio è molto utile in sede di debug, in particolare per quando riguarda i sistemi che si interfacciano a dispositivi hardware, come i sensori o il veicolo, i quali non sono sempre disponibili. Potremo infatti controllare il comportamento di alcuni componenti in determinate situazioni *off-line*, apportare modifiche, e verificarne immediatamente gli effetti senza dover necessariamente accedere al veicolo.

Anche in questo il caso il servizio farà uso delle interfacce di comunicazione dei vari componenti per leggerne i dati prodotti. Allo stesso tempo esso dovrà reimplementare le stesse interfacce, per poter simulare i servizi stessi in fase di playback.

2.3 Schema di controllo

Nel Paragrafo precedente abbiamo descritto quali sono i servizi necessari per un veicolo autonomo, almeno dal punto di vista funzionale. Abbiamo anche detto che essi si scambieranno informazioni attraverso un'apposita interfaccia, che sarà oggetto di studio del Paragrafo 3.4 del prossimo Capitolo. Il semplice scambio di informazioni non è comunque sufficiente per il corretto funzionamento di un tale sistema di servizi, infatti essi hanno anche necessità di *coordinarsi*, hanno bisogno di essere inizializzati

con le informazioni sulla missione corrente, devono ricevere il comando di inizio della missione, di arresto della missione, ecc. In questo Paragrafo descriveremo proprio l'importante schema di controllo dei servizi, cioè l'insieme di interazioni e servizi supplementari necessari a questo compito.

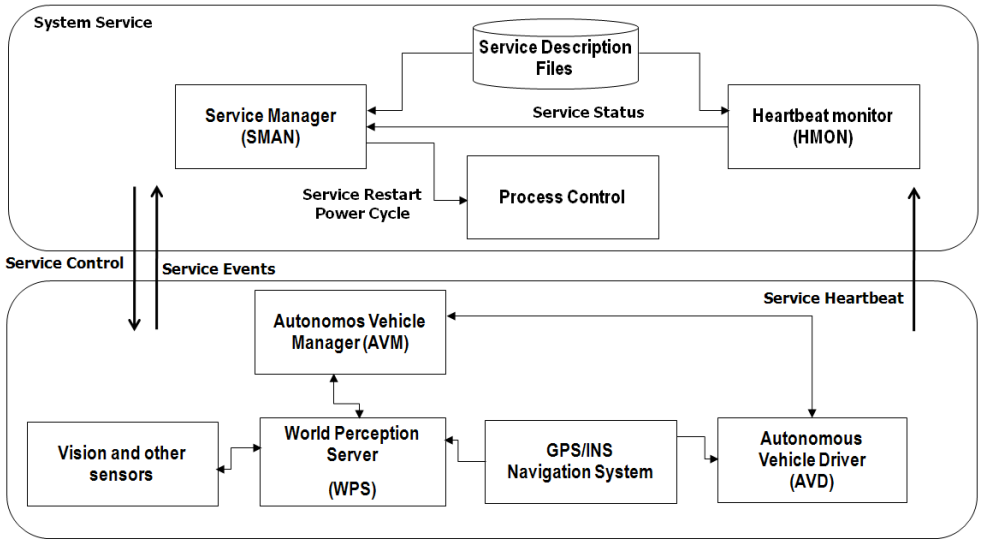


Figura 2.3: Esempio della struttura di controllo, con i relativi servizi coinvolti.

In Figura 2.3 è illustrato lo schema delle interazioni di controllo. Nella parte bassa sono stati riportati i principali servizi illustrati nel Paragrafo precedente, mentre nella parte superiore sono mostrati alcuni nuovi servizi e indicate alcune interazioni tra di essi. Il principio su cui si basa tutto lo schema è quello dell'*heartbeat*: ogni servizio deve comunicare *periodicamente* attraverso tutta la rete (broadcast) la propria presenza, il proprio stato e le informazioni necessarie affinché altri servizi possano raggiungerlo e attivare una comunicazione. Tipicamente questo messaggio conterrà quindi l'indirizzo IP e la porta del servizio, il suo nome unico e lo stato corrente. Sebbene il messaggio sia inoltrato a tutta la rete, è bene che esista un servizio appositamente progettato per raccogliere questi segnali, e quindi essere in grado di fornire una resoconto aggiornato della situazione attuale del sistema, che verrà chia-

mato *Heartbeat Monitor*. Al fianco di questo speciale servizio ne avremo un altro di grande importanza, che implementerà tutte le funzioni di gestione dei servizi, chiamato Service Manager: esso riceve l'elenco attuale dei componenti sulla rete proprio dall'Heartbeat Monitor, verifica che tutte le dipendenze sia soddisfatte, e quindi fa partire la missione.

È interessante notare come questo meccanismo permetta ai servizi di non necessitare della conoscenza reciproca *statica*: i vari servizi non hanno bisogno di sapere a priori su quale porta interrogare un altro componente del sistema, in quanto riceveranno queste informazioni grazie all'Heartbeat Monitor, che avrà raccolto tutti i dati forniti dai servizi stessi. Questo permette una grande flessibilità durante la fase di sviluppo, i servizi possono essere spostati da un sistema di elaborazione ad un altro, possono essere facilmente inglobati o separati da altri, è possibili implementare efficienti meccanismi di adattamento dinamico nel caso qualche servizio dovesse risultare irraggiungibile o non più funzionante. Nei prossimi Paragrafi vedremo meglio alcuni dettagli sul sistema ad Hearbeat, per il momento passiamo ad una breve carrellata di processi coinvolti nello schema di controllo:

2.3.1 Heartbeat Monitor

Come anticipato, il servizio Hearbeat Monitor ha il compito di restare in attesa dei messaggi inviati in modalità Broadcast o Multicast sulla rete. Il servizio offrirà, tramite l'interfaccia di comunicazione, la possibilità di collegarsi ad esso e di richiedere gli aggiornamenti sullo stato dei componenti del sistema, come ad esempio viene fatto da Service Manager, di cui discuteremo in seguito (Paragrafo 2.3.3).

2.3.2 Process Control

Questo servizio, come già il Playback Server, non si può definire strettamente necessario all'architettura di un veicolo autonomo. L'esperienza accumulata sui veicoli autonomi nell'arco di in questi anni però ha fatto emergere chiaramente la necessità di poter, in casi eccezionali, riavviare fisicamente gli elaboratori che ospitano i ser-

vizi. Come ogni software, anche i sevizi più importanti saranno affetti da bug, che in qualche caso potranno mandarli in blocco: in questi casi sarà compito del Process Control fornire un'interfaccia per il riavvio fisico del processori, tramite un apposito bus hardware, come ad esempio il CAN.

2.3.3 Service Manager

Veniamo infine alla descrizione del servizio più importante per lo schema di controllo, il Service Manager. Come si evince dal nome assegnatogli, esso ha il compito di gestire l'insieme dei servizi del veicolo. Le principali responsabilità solo le seguenti:

- *lettura dei dati missione*: dati come i waypoint, le informazioni sul tipo di strade da attraversare eventualmente disponibili, l'elenco dei componenti software che saranno coinvolti nella guida automatica (Autonomous Vehicle Manager, Autonomous Vehicle Driver, ecc.), i parametri di configurazione generali della gestione dei servizi (ad esempio i time out), ecc., devono essere noti staticamente al sistema, prima di iniziare la fase di guida automatica. Essi saranno immagazzinati in file di configurazione, e letti dal Service Manager all'avvio, in modo che possa gestire il coordinamento dei servizi. Notiamo che in questo file i servizi sono noti solo per il loro nome, ma non per l'indirizzo IP e la porta sulla quale rispondo, che verranno acquisiti dinamicamente tramite l'heartbeat.

- *controllo dipendenze*: il Service Manager apre una connessione verso l'Heartbeat Monitor, dal quale ricevere le informazioni sui servizi presenti in rete, quindi verifica che le rispettive dipendenze siano verificate.

- *inizializzazione dei servizi*: a questo punto il Service Manager è pronto per inizializzare il resto dei servizi presenti. Inviando un messaggio opportuno (che vedremo nei dettagli nel Paragrafo 3.5 del prossimo Capitolo), esso comunica l'elenco dei servizi presenti, non ché tutte le informazioni note staticamente sul percorso da attraversare, come la lista dei waypoints.

- *avvio ed arresto missione*: una volta inizializzati tutti i servizi, il Service Manager è pronto per inviare un comando di avvio delle operazioni a tutti i componenti. L'invio di questo messaggio è a sua volta comandato da un operatore umano, tramite l'interfaccia grafica. Allo stesso modo avviene l'arresto della modalità automatica, ancora con un apposito messaggio, il cui invio è conseguente ad una richiesta da parte dell'utilizzatore.

- *riavvio forzato*: nel caso l'Heartbeat Monitor dovesse comunicare che un servizio non sta più inviando il segnale periodico, il Service Manager può, ma non necessariamente deve, richiedere al Process Control di riavviarlo.

- *riconfigurazione servizi*: se non fosse più possibile riavviare forzatamente un servizio, o nel caso esso comunichi al Service Manager che non è più in grado di svolgere il suo compito (*service failure*), ad esempio perchè il sensore che controlla è danneggiato, il Service Manager invierà ai restanti un messaggio con la lista aggiornata dei servizi in funzione. Questo darà la possibilità ai vari componenti di riconfigurarsi *autonomamente*, quando questo è tecnicamente possibile, o di comunicare al Service Manager che in non è più in grado di continuare la missione (*fatal service falure*).

La comunicazione da parte del Service Manager verso i servizi avviene sia tramite l'interfaccia di comunicazione, ad esempio per la lettura dell'elenco dei servizi attivi dal Heartbeat Monitor o la scrittura del servizio da riavviare verso il Process Control, sia tramite l'interfaccia di controllo, che implementa i meccanismi di coordinamento e comando visti in precedenza, e che sarà oggetto di studio del prossimo Capitolo.

Capitolo 3

Le interfacce

Questo capitolo introduce le interfacce di comunicazione e controllo, che costituiscono l'aspetto caratterizzante dell'architettura proposta.

3.1 Comunicazione attraverso reti locali

Nel Capitolo precedente abbiamo visto quali sono i principali servizi che fanno parte dell'architettura, e come essi si basino sulle interfacce di comunicazione e controllo per interagire reciprocamente. Dedichiamo questo Capitolo alla descrizione dettagliata delle interfacce, illustrandone sia la codifica dei messaggi, sia le sequenze operative che definiscono le interazioni tra servizi.

Come già anticipato, il sistema è pensato per un veicolo autonomo dotato di elaboratori connessi tra loro attraverso una rete locale LAN. In questo contesto possiamo dire che lo scambio dei messaggi avverrà attraverso i seguenti protocolli:

Broadcast/Multicast : questo protocollo è utilizzato per la trasmissione degli heartbeat. In particolare può essere utile adottare indirizzi della classe Multicast quando si vuole realizzare una comunicazione in broadcast internamente a gruppi di servizi, i quali potrebbero essere gestiti da propri Heartbeat Monitor

e Service Manager. Questi, a loro volta, potrebbero essere coordinati da un supervisore di più alto livello, in uno schema piramidale che permetterebbe una elevata scalabilità del sistema.

TCP/IP : tutte le informazioni prodotte dai servizi e i messaggi di controllo vengono scambiate mediante questo protocollo. Per prima cosa verrà aperta una *socket* all'indirizzo e la porta del servizio con cui si vuole instaurare una comunicazione, dopo di chè, non appena ricevuto il comando di start dal Service Manager, inizierà la lettura e scrittura dati.

XML : per la codifica dei messaggi stessi si è optato per l'utilizzo di stringhe in formato XML. Esso garantisce un'elevata flessibilità nella modifica delle strutture dati, e la possibilità di mantenere compatibilità con le vecchie versioni. Inoltre, esistono numerose librerie software, per quasi ogni linguaggio e sistema operativo, che possono facilitare il lavoro dello sviluppatore nella decodifica di stringhe XML da e verso strutture dati binarie.

Esistono molti sistemi e protocolli per la comunicazione attraverso rete locale, oltre a quello che andremo a descrivere. In particolare vale forse la pena ricordare quello utilizzato dal veicolo TerraMax alla Grand Challenge 2004 [1]: si trattava si un sistema a *condivisione di memoria* sviluppato dal NIST, il quale prevedeva la possibilità, per processi distinti sulla rete, di accedere ad una zona di memoria comune, contenente i dati da aggiornare. In questo contesto i sensori erano tra gli elementi *scriventi*, mentre i componenti di alto livello come l'Autonomous Vehicle Manager erano tra i *lettori*. Il sistema era certamente molto interessante, e con grandi potenzialità, ma alla fine si è rilevato non adatto allo scopo. La sua struttura era troppo rigida e complessa, mentre, per un progetto di quel tipo, la flessibilità e la semplicità di implementazione erano caratteristiche dalle quali non si poteva prescindere.

3.2 Sistemi di riferimento

Nei prossimi paragrafi vedremo nel dettaglio alcuni messaggi scambiati dai servizi. Essi contengono, tra le altre cose, la codifica degli elementi del mondo fisico percepiti dai sensori. Questo tipo di informazioni richiede necessariamente la definizione di un sistema di riferimento, nel quale questi elementi verrano posti, localizzati e descritti. Naturalmente ogni implementazione può presentare propri sistemi di riferimento, per i quali l'architettura e le interfacce illustrate continuano a valere.

Il sistema di riferimento solidale con il veicolo scelto è lo $x-y-z$ canonico, la cui origine viene posta nel punto più avanzato del veicolo, con la x nella direzione del moto, e la z diretta verso l'alto. Per quando riguarda il sistema di riferimento globale, esso utilizza il classico latitudine/longitudine per la geo-referenziazione, mentre usa un Nord/Est/Basso come sistema di riferimento Euclideo per il movimento effettivo del veicolo.

Tutte le misure saranno espresse in *cm*, questo per mantenere i dati in formato intero, più veloce da essere convertito da stringa a binario.

3.3 Codifica di base dei messaggi

Nei Paragrafi 3.4.2 e 3.5.3 vedremo i dettagli della codifica XML dei messaggi per entrambe le interfacce. Esse sono caratterizzate da alcuni elementi comini, che si ripetono in praticamente ogni messaggio, vediamoli:

```
<Message timeStampUTC="1170167040218">
   <Src serviceID="VISION"/>
   <Dst serviceID="WPS"/>
   <Body>
   ...
   </Body>
</Mssage>
```

In altre parole ogni messaggio è caratterizzato da un nome unico, il time stamp (nel nostro caso millisecondi dal 1 Gennaio 1970), il mittente, il destinatario, ed il corpo delle informazioni trasmesse. È proprio nella parte denominata "Body" che i vari messaggi si differenzieranno.

Fa eccezione a questa struttura il messaggio Heartbeat, che non contiene il destinatario, essendo inviato in broadcast su tutta la rete.

3.4 Interfaccia di comunicazione

L'interfaccia di comunicazione definisce le modalità con cui due servizi possono scambiare informazioni. Le informazioni scambiate non riguardano il coordinamento, né il comando dei servizi stessi, ma semplicemente dati utili per le elaborazioni specifiche dei singoli componenti. Ad esempio, il World Perseption Server riceverà attraverso i meccanismi dell'interfaccia di comunicazione la lista degli ostacoli attuale, i confini della carreggiata, ma anche la velocità e la posizione GPS del veicolo. Allo stesso modo l'Autonomous Vehicle Manager riceverà dal World Perseption Server un messaggio contenente tutte le informazioni di percezione del veicolo, fuse tra loro, a formare un'immagine complessiva del mondo circostante il veicolo. Attraverso questa interfaccia non riceverà invece i comandi di avvio ed arresto della missione, il messaggio di inizializzazione, e non invierà tramite di essa il suo heartbeat.

Andiamo ad illustrare per prime le così dette *sequenze operative*, cioè una descrizione delle sequenze di messaggi scambiati per realizzare una comunicazione, poi vedremo la *codifica dei messaggi*, cioè la struttura delle stringhe XML che formano i messaggi stessi.

3.4.1 Le sequenze operative

L'interfaccia di comunicazione prevede due tipi di sequenze operative:

Subscribe : questa modalità viene utilizzata da un servizio per richiedere l'invio *continuativo*, a flusso, di un particolare tipo di dato da parte di un altro ser-

vizio. L'azione di inviare il dato richiesto verrà chiamata *publishing*. La sequenza operativa è la seguente: 1) il servizio richiedente apre una socket su un secondo servizio, ed invia un messaggio Subscribe, in cui indica il tipo di dato di cui ha bisogno; 2) il ricevente verifica di essere in grado di fornire il dato, e risponde affermativamente o negativamente alla richiesta tramite un messaggio SubscribeResponse; 3) il ricevente inizierà a scrivere sulla socket i dati non appena avrà ricevuto il messaggio di Start da parte del Service Manager. In Figura 3.1 una schematizzazione di questa sequenza.

Request/Response : quando un servizio non ha bisogno di ricevere un flusso continuativo di dati, ma ha necessità di un singolo dato in modo episodico, aperiodico, allora utilizzerà la modalità Request/Response: 1) il servizio richiedente apre una socket su un secondo servizio, ed invia un messaggio Request, in cui indica il tipo di dato di cui ha bisogno; 2) il ricevente verifica di essere in grado di fornire il dato richiesto, e risponderà con un messaggio di tipo Response specifico, contenente le informazioni desiderate. In Figura 3.2 una schematizzazione di questa sequenza.

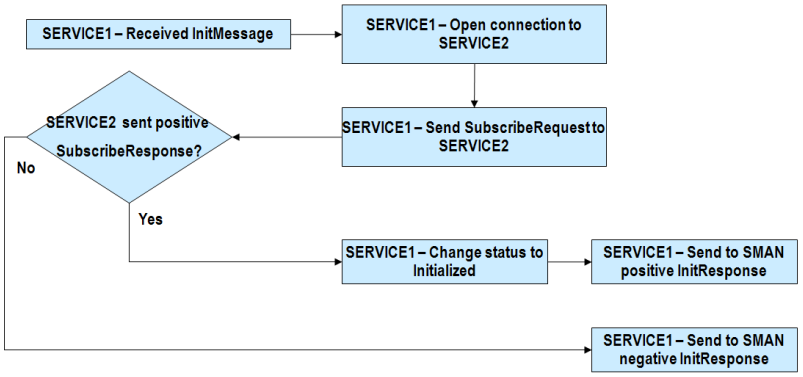


Figura 3.1: Schematizzazione della sequenza operativa Subscribe.

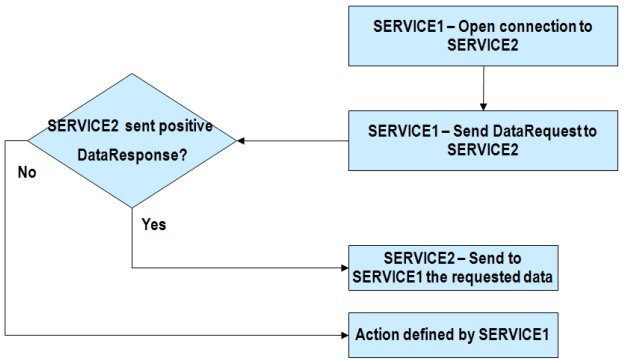


Figura 3.2: Schematizzazione della struttura ad interfacce di un servizio.

3.4.2 Codifica dei messaggi

Nei prossimi Paragrafi vederemo alcuni esempi di come appaiono i messaggi inviati dai sensori per comunicare gli elementi del mondo fisico che hanno percepito. Vedremo poi come sono rappresentati i messaggi inviati dal Navigation System per comunicare la posizione del veicolo, e i dati di assetto. Porteremo un esempio del messaggio utilizzato dal World Perseption Server per rappresentare lo stato attuale del mondo, ed infine vedremo il messaggio inviato dall'Autonomous Vehicle Manager per fornire aggiornamenti sullo stato della guida automatica. Iniziamo la carrellata di messaggi con quelli di Subscribe, Request e corrispondenti risposte.

Messaggi di Subscribe. Quando un servizio vuole richiedere ad un altro componente del sistema l'invio continuativo di dati utilizzerà il messaggio Subscribe, indicando in esso il tipo di informazione di cui ha bisogno:

```
<Subscribe requestID="12" timeStampUTC="1163409600"
  topic="SENSOR_UPDATE">
   <Src serviceID="WPS"/>
   <Dst serviceID="VISON"/>
</Subscribe>
```

Il ricevente, dopo aver verificato la possibilità di rispondere positivamente o negativamente alla Subscribe, invierà il seguente messaggio, impostando opportunamente il campo result:

```
<SubscribeResponse requestID="12" result="true"
 subscriptionID="1" timeStampUTC="1163409601">
   <Src serviceID="VISION"/>
   <Dst serviceID="WPS"/>
</SubscribeResponse>
```

Messaggi di Request/Response. Quando un servizio vuole richiedere un dato in modo episodico, non continuativo, utilizzerà il messaggio Request, indicando nel campo "topic" il tipo di informazione di cui ha necessità.

```
<Request requestID="12" timeStampUTC="1163409600"
  topic="SCAN_FOR_OBSTACLES">
   <Src serviceID="WPS"/>
   <Dst serviceID="VISON"/>
</Request>
```

Il ricevente verificherà di essere in grado di fornire il dato, ed eventualmente lo includerà nella risposta al messaggio di Request:

```
<Response requestID="12" result="true"
 timeStampUTC="1163409600"
  topic="SCAN_FOR_OBSTACLES">
   <Src serviceID="WPS"/>
   <Dst serviceID="VISON"/>
   <Body>..... </Body>
</Request>
```

Messaggi per sensori: Ostacoli. Un ostacolo è definito come un poligono chiuso sul piano $x-y$ in cui si muove il veicolo, dotato di un'altezza complessiva e di alcuni attributi: identificativo unico (objectID), età (objectAge), sensore che lo ha creato (sensorID) e velocità (velX, velY).

```
<ObstacleDetected timeStampUTC="1170167040218">
   <Src serviceID="VISION"/>
   <Dst serviceID="WPS"/>
   <Object objectID="1" velX="301" velY="301"
         height="101"  objectAge=1  sensorID="VISION>
        <Point x="50000" y="1500"/>
        <Point x="50000" y="1000"/>
        <Point x="50000" y="500"/>
        <Point x="50000" y="0"/>
   </Object>
   <Object objectID="2" velX="401" velY="301"
        height="201"  objectAge=1 sensorID="VISION>
        <Point x="50000" y="1501"/>
        <Point x="50000" y="1001"/>
        <Point x="50000" y="501"/>
        <Point x="50000" y="1"/>
   </Object>
   <Object objectID="3" velX="501" velY="301"
        height="301"  objectAge=1 sensorID="VISION>
        <Point x="50000" y="1502"/>
        <Point x="50000" y="1002"/>
        <Point x="50000" y="502"/>
        <Point x="50000" y="2"/>
   </Object>
</ObstacleDetected>
```

Messaggi per sensori: linee di demarcazione corsie e percorso. Spesso accade che non sia possibile identificare univocamente il percorso da seguire semplicemente osservando lo spazio di fronte al veicolo, in quando esso risulto definito in gran parte dalle sue intenzioni di movimento. Immaginiamo una strada a molte corsie, in funzione della posizione del veicolo una di esse sarà la corsia di marcia, mentre le altre saranno corsie laterali dove potremmo eventualmente spostarci. Tutti questi elementi sono contenuti all'interno della così detta carreggiata, che può avere confini anche difficilmente identificabili: marciapiede, erba, cambio di superficie, o semplicemente l'assenza di linee di demarcazione delle corsie. In questo scenario è molto più semplice limitarsi all'individuazione delle linee che demarcano le corsie, e lasciare al sistema di alto livello stabilire quale di esse sia la corrente, quale sia la migliore da percorrere, e quali siano invece i limiti della carreggiata da non oltrepassare.

Le linee di demarcazione delle corsie sono definite come delle spezzate nel piano $x - y$, e caratterizzate da un colore: bianca continua, gialla continua, bianca tratteggiata, doppia gialla continua, sconosciuta.

```
<LaneUpdate timeStampUTC="1163409201“>
  <Src serviceID="VISION"/>
  <Dst serviceID="WPS"/>
   <RoadEdge color=“double_yellow”>
        <Point x="-10123" y="100"/>
        <Point x="-10143" y="1000"/>
        <Point x="-10150" y="10000"/>
   </RoadEdge>
   <RoadEdge color=“broken_white”>
        <Point x="-5123" y="100"/>
        <Point x="-5143" y="1000"/>
        <Point x="-5150" y="10000"/>
   </RoadEdge>
   <RoadEdge color=“solid_white”>
        <Point x="10123" y="100"/>
```

```
        <Point x="10143" y="1000"/>
        <Point x="10150" y="10000"/>
   </RoadEdge>
</LaneUpdate>
```

Quando le linee di demarcazione non sono presenti in nessuna regione dell'immagine, saremo obbligati a ricercare i confini della carreggiata, per quanto estremamente vaga e soggettiva sia la sua definizione. L'interfaccia prevede una codifica che altro non sarà che una versione semplificata del LaneUpdate, dove sono stati eliminati gli attributi di colore, e dove il numero di spezzare è ridotto a due, in quanto sufficienti a rappresentare i confini dello spazio percorribile.

```
<PathUpdate timeStampUTC="1163409201">
  <Src serviceID="VISION"/>
  <Dst serviceID="WPS"/>
   <RoadEdge>
        <Point x="-10123" y="100"/>
        <Point x="-10143" y="1000"/>
        <Point x="-10150" y="10000"/>
   </RoadEdge>
   <RoadEdge>
        <Point x="-5123" y="100"/>
        <Point x="-5143" y="1000"/>
        <Point x="-5150" y="10000"/>
   </RoadEdge>
</PathUpdate>
```

Messaggi per sensori: linea di stop. Le linee di stop sono semplicemente definite come una coppia di punti nel piano $x-y$, che rappresentano gli estremi della linea. Sfortunatamente è difficile, in generale, distinguere tra una linea di stop ed una qualunque linea bianca trasversale disegnata sull'asfalto. Per questa ragione il protocollo

prevede la possibilità di inviare una serie di linee trasversali, ogni una delle quali potenzialmente rappresenta una linea di stop. Sarà compito dell'alto livello confrontare la posizione delle linee fornite con le informazioni disponibili sulla struttura della strada, per capire quale effettivamente sia la demarcazione dell'incrocio che stiamo cercando.

La linea di stop è caratterizzata, oltre che dalla posizione, anche dalla sua larghezza.

```
<StopLines timeStampUTC="1170167040218">
   <Src serviceID="VISION"/>
   <Dst serviceID="WPS"/>
   <LineMarking width="100">
       <Point x="300" y="1500"/>
       <Point x="300" y="-1500"/>
   </LineMarking>
   <LineMarking width="101">
       <Point x="300" y="1501"/>
       <Point x="300" y="-1501"/>
   </LineMarking>
</StopLines>
```

Messaggi per sensori: INS, GPS, assetto veicolo e posizione. L'interfaccia di comunicazione prevede che il sistema di navigazione comunichi l'assetto e la posizione del veicolo tramite due distinti messaggi: una chiamato NavAttitude, contenente assetto e velocità, ed uno chiamato NavGeodetic, contenente la posizione del veicolo nel mondo.

Il messaggio NavAttitude contiene: velocità lungo i 3 assi del sistema di riferimento $x-y-z$ scelto (nel nostro caso Nord/Est/Basso), orientazione in radianti attorno ai tre assi (pitch, roll, heading), la velocità di rotazione attorno ai tre assi (Rate) ed infine l'accelerazione lungo i tre assi (Accel).

```
<NavAttitude timeStampUTC="1163409185">
```

```
   <Src serviceID="NAV" />
   <Dst serviceID="WPS" />
   <Vel foward="40" north="30" east="20" />
   <Orientation pitch="1.5" roll="0.0" heading="4.5" />
   <Rate x="0.0" y="0.0" z=""/>
   <Accel x="0.0" y="0.0" z="9.8"/>
 </NavAttitude>
```

Il messaggio NavGeodetic contiene: latitudine, longitudine e posizione nel sistema di riferimento $x-y-z$ scelto.

```
<NavGeodetic timeStampUTC="1163409185">
   <Src serviceID="NAV" />
   <Dst serviceID="WPS" />
   <Coordinate latitude="39.12" longitude="-141.23" />
   <Pos north="5000" east="2000" zone="19" />
</NavGeodetic>
```

Già in questi semplici messaggi è possibile notare la semplicità e la flessibilità che si ottengono dall'uso di stringhe XML. La notazione non è posizionale, ma ogni campo è chiaramente indicato dal proprio nome, in modo che non sia possibile scambiare due dati distinti invertendone l'ordine di lettura. Allo stesso modo l'utente è facilitato nella lettura di queste strutture in fase di debug, e non necessità di ricordare la posizione, la dimensione ed il tipo di dato che sta cercando, come farebbe con una struttura di tipo binario.

Messaggio del World Perseption Server: aggiornamento della mappa del mondo. Vediamo infine un importante tipo di messaggio, il WorldMapUpdate. Esso contiene una rappresentazione complessiva del mondo fisico che circonda il veicolo, così come viene percepito dai sensori. Il servizio World Perseption Server ha il compito di raccogliere queste informazioni, e fonderle assieme par ottenere un'unica immagine della percezione sensoriale del veicolo. Questo tipo di messaggio viene

inviato periodicamente all'Autonomous Vehicle Manager, che né ha fatto richiesta tramite il meccanismo Subscribe, per essere utilizzato nella guida automatica del veicolo.

Il messaggio può essere in generale molto grande, dovendo contenere informazioni provenienti da molti sensori, i quali, a loro volta, possono produrre numerosi dati ad ogni ciclo di scansione. Esso sarà formato da vari blocchi, alcuni dei quali facoltativi: informazione di navigazione (velocità, posizione, ecc.), una lista di ostacoli provenienti dai vari sensori, una lista delle linee di demarcazione corsie, una lista dei confini della carreggiata, una lista delle possibili stop line. Notiamo come in alcuni di questi blocchi è riportato il sensore che ha generato l'informazione, mentre in altri è presente soltanto la dicitura "FUSED", ad indicare che esso è stato ottenuto dalla fusione tra informazioni sensoriali provenienti da vari sensori.

```
<WorldMapUpdate timeStampUTC="1193710141046">
   <Src serviceID="WPS" />
   <Dst serviceID="AVM" />
   <Vehicle>
     <Pos zone="11" north="382588192" east="46710037" />
     <Vel foward="-1" north="0" east="1" up="0" />
     <Orientation heading="4.4408597946167" />
   </Vehicle>
   <Obstacles>
      <Object objectID="75" velX="-4" velY="6"
      width="-2082" length="120" objectAge="1284"
      sensorID="LIDAR">
        <Point x="-2156" y="2234" />
        <Point x="-2011" y="2264" />
      </Object>
      <Object objectID="4444" velX="0" velY="0"
      height="158" objectAge="1" sensorID="FUSED">
        <Point x="1437" y="-411" />
```

```
      <Point x="1455" y="-409" />
      <Point x="1324" y="-345" />
      <Point x="1437" y="-411" />
    </Object>
  </Obstacles>
  <LaneBoundaries>
    <PreciseLaneBoundaries>
       <LaneBoundary pos="Left"
       boundary_type="double_yellow"
       sensorID="VISION">
         <Point x="69" y="179" />
         <Point x="169" y="189" />
         <Point x="279" y="200" />
       </LaneBoundary>
       <LaneBoundary pos="Right"
       boundary_type="solid_white"
       sensorID="VISION">
         <Point x="570" y="-140" />
         <Point x="629" y="-120" />
         <Point x="819" y="-80" />
         <Point x="869" y="-70" />
       </LaneBoundary>
    </PreciseLaneBoundaries>
  </LaneBoundaries>
  <LatLineMarkings>
     <LineMarking width="100" sensorID="VISION">
       <Point x="300" y="1500"/>
       <Point x="300" y="-1500"/>
     </LineMarking>
  </LatLineMarkings>
```

```
</WorldMapUpdate>
```

Messaggio dell'Autonomous Vehicle Manager: aggiornamento sullo stato di marcia. L'ultimo messaggio dell'interfaccia di controllo che andiamo ad illustrare è il RaceContextUpdate, inviato dall'Autonomous Vehicle Manager. Il contenuto principale è un aggiornamento sullo stato del veicolo dal punto di vista della guida automatica e della missione: qui troveremo informazioni come i prossimi waypoint pianificati, un flag indicante se siamo fermi ad un incrocio, se stiamo approcciando un incrocio ma non ci siamo ancora fermati, se siamo fermi dietro ad un altro veicolo, se stiamo eseguendo una manovra di aggiramento ostacolo, ecc. Il messaggio viene inviato a tutti i servizi che ne fanno richiesta, ed in particolare ricordiamo: i sensori, i quali utilizzano queste informazioni per attivare l'esecuzione di particolari algoritmi, come la rilevazione degli ostacoli agli incroci, la rilevazione delle linee di stop, o per stimare la traiettoria del veicolo ed eseguire una più accurata rilevazione delle linee di carreggiata; l'Autonompus Vehicle Driver, che invece utilizzerà queste informazioni per muovere il veicolo attraverso waypoint attesi, alla velocità richiesta.

```
<RaceContextUpdate timeStampUTC="1193710141046">
   <Src serviceID="WPS" />
   <Dst serviceID="AVM" />
   <RaceContext raceState="Running" lastWaypointIndex="1"
   lastIssuedWaypointID="1" lastIssuedWaypointSpeed="2"
   approachingStopline="False" atStopline="False"
   atCheckpoint="False" stopped="True" stopping="False"
   following="False" waitingBehindObstacle="False"
   currentTurnIndicator="None" obstacleAvoidance="False"/>
   <PlannedState>
     <NextWaypointList>
       <Waypoint waypointID="20.2.8" />
       <Waypoint waypointID="20.2.9" />
       <Waypoint waypointID="20.2.10" />
```

```
    </NextWaypointList>
  <Vel foward="10">
</RaceContextUpdate>
```

3.5 Interfaccia di controllo

L'interfaccia di controllo definisce le modalità con cui il Service Manager sincronizza, coordina, avvia, arresta e configura il lavoro svolto dai servizi del sistema. Questo tipo di interazioni non coinvolgono lo scambio di dati utili per l'elaborazione, ma ha il solo scopo di gestire il lavoro dei vari componenti, visti come *gruppo distribuito* di processi di elaborazione.

3.5.1 Stati possibili dei servizi

Per meglio descrivere il funzionamento di questa interfaccia è necessario introdurre il concetto di *stato* dei servizi. I servizi, durante l'intero loro ciclo di vita, possono trovarsi in uno dei seguenti stati:

- *Uninitialized*: questo è lo stato che assume un servizio non appena entra in esecuzione, in questa fase esso non è ancora venuto in contatto con nessun altro servizio, non conosce nessun dettaglio della missione corrente, né come potrebbe accedere agli altri componenti del sistema.

- *Initialized*: in questo stato il servizio ha ricevuto il messaggio di inizializzazione dal Service Manager, ha verificato la presenza dei servizi di cui ha bisogno per l'elaborazione, ed ha eseguito con successo le richieste di Subscribe.

- *Running*: il servizio ha ricevuto il messaggio di avvio dal Service Manager, sta inviando i dati che produce ai servizi che ne avevano fatto richiesta, legge da altri servizi i dati di cui ha necessitá;

- *Stopped*: il servizio ha ricevuto il segnale di Stop dal Service Manager, ha quindi interrotto l'invio dati e ha chiuso tutte le connessioni, sia in lettura che in scrittura.

- *Failure*: in questo caso il servizio comunica al sistema che non è più in grado di eseguire le operazioni che gli competono. Spetterà al Service Manager agire di conseguenza.

Lo stato del servizio viene comunicato periodicamente attraverso l'heartbeat.

3.5.2 Le sequenze operative

I meccanismi base previsti da questa interfaccia solo:

Cattura degli heartbeat e configurazione missione : in questa fase l'Heartbeat Monitor raccoglie le informazioni inviate dai servizi, contenenti il loro stato e l'indirizzo e porta alla quale possono essere raggiunti. L'insieme di queste informazioni viene inviato, tramite l'interfaccia di comunicazione, al Service Manager nel momento in cui ne fa richiesta. Quest'ultimo avrà caricato da file i parametri di configurazione della missione, includendo: punto di partenza e punto di arrivo, waypoint, caratteristiche della strada, limiti di velocità quando noti, posizione degli incroci quando nota, elenco dei servizi necessari per questa missione. In questa fare il Service Manager non conosce altro che il nome dei servizi che faranno parte del sistema di guida automatico, che associerà con le le informazioni inviategli dall'Heartbeat Monitor, per ricavarne tutte le altre informazioni.

In questa fase tutti i servizi sono nello stato "Uninitialized", eccetto il Service Manager e l'Heartbeat Monitor, che sono costantemente in "Running". In Figura 3.3 la schematizzazione di questa sequenza.

Inizializzazione ed avvio dei servizi : una volta che il Service Manager ha raccolto tutte le informazioni necessarie per la missione passerà all'inizializzazione

dei servizi. Questo avviene inviando ad ogni uno di essi un messaggio contenente l'elenco dei servizi attualmente attivi, più l'elenco dei waypoint e tutte le informazioni sulla strada e la missione disponibili. In questo modo i servizi, che inizialmente non conoscono gli altri componenti sulla rete ne' sanno come accedervi, potranno configurarsi in base all'insieme dei servizi presenti. In particolare essi andranno ad aprire una connessione su quelli da cui desiderano ricevere informazioni per svolgere le loro elaborazioni: ad esempio il World Perseption Manager si connetterà ai vari sensori e richiederà loro le informazioni sensoriali che possono produrre (Subscribe), allo stesso modo l'Autonomous Vehicle Manager richiederà al World Perseption Server la mappa del mondo aggiornata, ecc. Notiamo come il tutto possa avvenire in modo dinamico, in funzione dei servizi effettivamente presenti sulla rete, senza una precedente conoscenza reciproca. Una volta che tutte le connessioni sono aperte, e le Subscribe sono state effettuate con successo, i servizi cambiano il loro stato in "Initalized".

A questo punto il Service Manager è pronto inviare un messaggio di Start non appena questo verrà richiesto dall'utente, attraverso il servizio User Interface. La ricezione del messaggio di Start da parte di un servizio provoca l'inizio della scrittura dei dati per i quali ha ricevuto una Subscribe, la lettura dei dati per i quali ha inviato una Subscribe, e la modifica dello stato da "Initialized" a "Running". In Figura 3.4 la schematizzazione di questa sequenza.

Arresto dei servizi : quando il Service Manager riceve il comando di arrestare la missione da parte dell'utente, viene inviato un apposito messaggio a tutti i servizi. Alla ricezione di tale messaggio ogni servizio arresterà l'invio di dati per i quali ha accettato una richiesta di Subscribe, e chiuderà tutte le connessioni, sia in lettura che in scrittura. Infine porta il proprio stato da "Running" a "Stopped". In Figura 3.5 la schematizzazione di questa sequenza.

Riavvio forzato di un servizio : quando un servizio non invia piè l'heartbeat, o ne sta inviando uno contenente lo stato "Failure", il Service Manager potrebbe de-

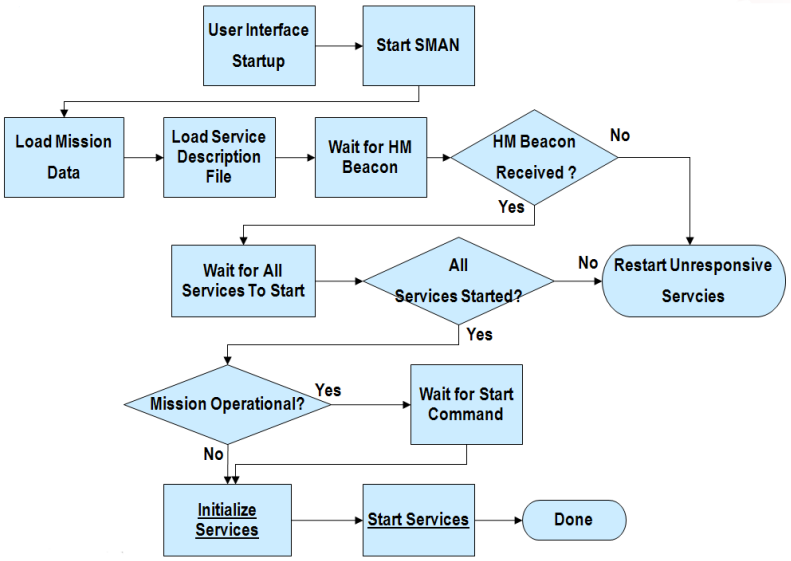


Figura 3.3: Schematizzazione della fase di cattura degli heartbeat e configurazione missione

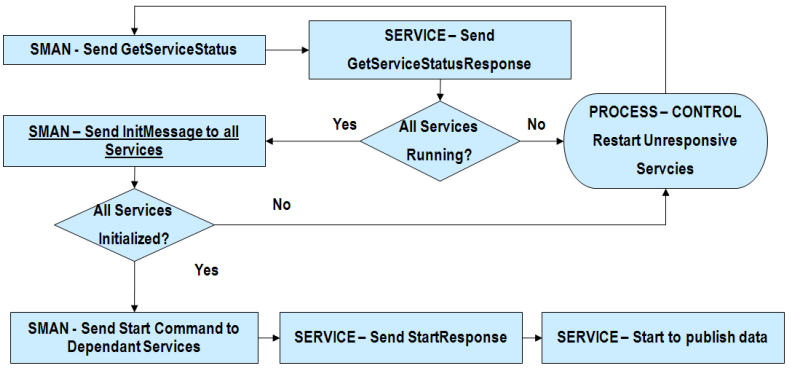


Figura 3.4: Schematizzazione della fase si inizializzazione ed avvio dei servizi.

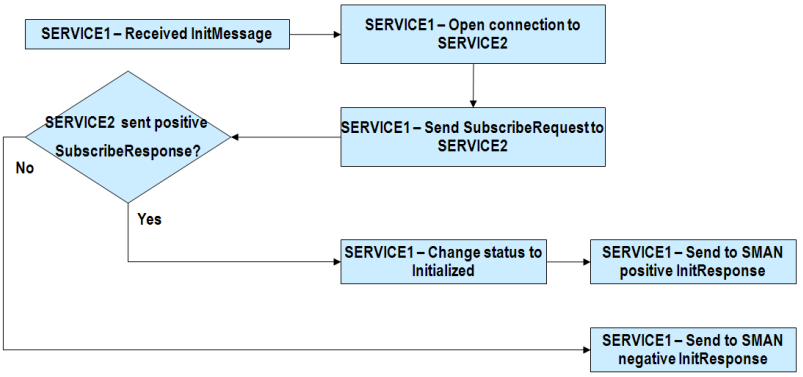


Figura 3.5: Schematizzazione della fase si arresto della missione

cidere di riavviare forzatamente l'elaboratore che lo ospita, sperando in questo modo di superare il problema occorso. Per fare questo si avvale dell'interfaccia fornita dal Process Control, che comanda qualche meccanismo hardware opportuno, come ad esempio il bus CAN. In Figura 3.6 la schematizzazione di questa sequenza.

Nel caso non fosse possibile riavviare il servizio con successo, il Service Manager attuerà una procedura per verificare se la missione può continuare lo stesso. Se così fosse, esso andrebbe a comunicare ai rimanenti servizi la perdita di uno dei componenti del sistema, attraverso un apposito messaggio che vedremo in dettaglio successivamente. A questo punto i servizi possono decidere quali sono le operazioni da intraprendere per gestire la situazione. Potrà anche accadere che qualche servizio non sia più in grado di proseguire il lavoro, nel qual caso comunicherà il proprio arresto mutando lo stato da "Running" a "Failed". Nuovamente, il Service Manager verificherà la possibilità di proseguire la missione, ed eventualmente procederà all'invio di un nuovo messaggio di aggiornamento. Nel caso venga a mancare un servizio necessario, la missione verrà abortita.

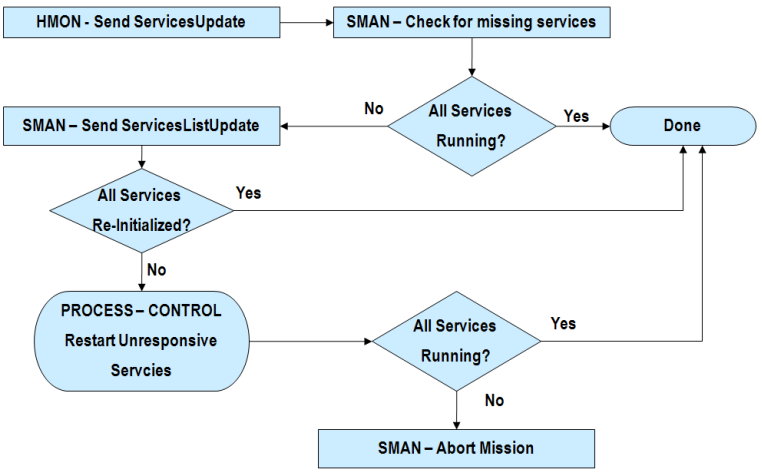


Figura 3.6: Schematizzazione della fase di riavvio forzato di un servizio.

3.5.3 Codifica dei messaggi

Chiudiamo questo Capitolo andando a descrivere la codifica dei principali messaggi utilizzati nell'interfaccia di controllo.

Messaggio Heartbeat. Questo è il messaggio inviato da tutti i servizi per comunicare di essere ancora in esecuzione, assieme alle seguenti informazioni: stato del servizio, indirizzo IP e porta al quale può essere raggiunto.

```
<Heartbeat serviceState="Uninitialized"
 timeStampUTC=1196347114574
 ipAddress="192.168.50.51"port="8080">
   <Src serviceID="VISIONE"/>
</Heartbeat>
```

Messaggio di inizializzazione. Il messaggio inviato dal Service Manager per inizializzare i servizi è il seguente:

```
<InitializeService requestID="20"
 timeStampUTC="1192528208781"
   <Src serviceID="SMAN" />
   <Dst serviceID="VISION"/>
     <Config>
        <Services>
          <Service name="SMAN" ipAddress="192.168.50.56"
           port="3005" />
          <Service name="HMON"ipAddress="192.168.50.56"
           port="3002"/>
          <Service name="AVM" ...... />
                ...
        </Services>
        <RNDF.... road information and missio data />
     </Config>
</InitializeService>
```

Esso contiene la lista dei servizi attualmente attivi, l'indirizzo e la porta su cui raggiungerli, più le informazioni disponibili su missione e strade che andremo a percorrere.

Messaggio di avvio missione. Questo messaggio viene inviato dal Service Manager per comunicare che è possibile avviare la missione. Alla ricezione i servizi inizieranno a fornire i dati richiesti, ed inizieranno a leggere i dati di cui hanno fatto domanda ad altri servizi.

```
<StartService requestID="2"
 timeStampUTC="1163409190">
   <Src serviceID="SMAN"/>
   <Dst serviceID="VISION"/>
</StartService>
```

Messaggio di arresto missione. Questo messaggio viene inviato dal Service Manager per comunicare che è necessario arrestare la missione. Tutte le connessioni verrano chiuse, ed il sistema si porterà nello stato di "Stopped".

```
<StopService requestID="4"
 timeStampUTC="1163409510">
   <Src serviceID="SMAN"/>
   <Dst serviceID="VISION/>
</StopService>
```

Messaggio di aggiornamento lista dei servizi. Con questo messaggio il Service Manager comunica il mutato assetto dei servizi. Ecco contiene la lista aggiornata dei servizi, con relativi indirizzi e porte.

```
<ServiceListUpdate requestID="20"
 timeStampUTC="1192528208781"
   <Src serviceID="SMAN"/>
   <Dst serviceID="VISION"/>
      <Services>
        <Service name="SMAN" ipAddress="192.168.50.56"
         port="3005" />
        <Service name="HMON" ipAddress="192.168.50.56"
         port="3002"/>
        <Service name="AVM" ...... />
                ...
      </Services>
</ServiceListUpdate>
```

Capitolo 4

Alcuni sistemi di visione artificiale per veicoli autonomi

In questo capitolo illustriamo alcuni sistemi di visione artificiale sviluppati durante il corso di Dottorato, ed installati su veicoli autonomi.

4.1 Rilevazione della carreggiata in ambienti non strutturati

Come abbiamo visto nel Capitolo precedente, uno dei possibili compiti a cui i sensori di un veicolo autonomo sono chiamati è la rilevazione della carreggiata quando siamo in ambienti non strutturati, dove cioè non esista una segnaletica orizzontale che demarchi i confini del percorso. Per le Grand Challenge 2004 e 2005 sono stati sviluppati dal Laboratorio di Visione Artificiale dell'Università di Parma due sistemi di questo tipo, con caratteristiche e filosofie piuttosto diverse. Vediamone una breve descrizione.

4.1.1 Rilevazione della carreggiata con agenti

Questo sistema è stato sviluppato nel 2004, e si basa su una tecnica di ottimizzazione ad agenti chiamata Ant Colony Optimizations [8] [9]. Le ipotesi su cui poggia l'algoritmo sono due: 1) è possibile individuare i confini della carreggiata come due curve sull'immagine, che collegano il bordo inferiore delle immagini stesse alla linea dell'orizzonte; 2) tali linee si possono ottenere muovendosi sull'immagine alla ricerca delle più elevate alle discontinuità delle proprietà dei pixel, assumendo che maggiore è la discontinuità, maggiore è la probabilità che quel pixel corrisponda ad un punto di passaggio tra strada e fuori strada.

Sulla base di queste ipotesi, l'algoritmo è sintetizzabile in questo modo:

- le discontinuità vengono messe in evidenza tramite un'operazione di derivazione sull'immagine, basata sui canali RGB.
- una coppia di colonie di agenti ha il compito muoversi, rispettivamente sulla parte destra e sinistra dell'immagine, alla ricerca del percorso che unisce il bordo inferiore dell'immagine con la linea corrispondente all'orizzonte, sovrapponendosi il più possibile alle discontinuità.

Il vantaggio nell'uso di una colonia di agenti che implementa la tecnica dell'Ant Colony Optimizations risiede nella possibilità di ricavare delle curve che siano *globalmente* ottime dal punto di vista della sovrapposizione con le discontinuità, riducendo il rischio di cadere in minimi locali causati da forti discontinuità puntuali.

In Appendice A vengono illustrati i dettagli implementativi di questo sistema.

4.1.2 Rilevazione della carreggiata tramite segmentazione

In occasione della Grand Challenge 2005 si è deciso di rivedere radicalmente l'approccio utilizzato l'anno precedente per il sistema di rilevazione della carreggiata.

La difficoltá principale nell'individuazione della strada in ambienti non strutturati é la definizione stessa di cosa sia una strada. Quando non esiste una segnaletica specifica, che univocamente definisca la carreggiata, la sua definizione includerá

necessariamente concetti qualitativi, come la forma, la percorribilitá del terreno, la direzione rispetto al movimento del veicolo. Nell'approccio al problema adottato nel 2004 siamo in effetti caduti nell'errore di dare una definizione eccessivamente precisa di cosa fosse una strada. Esso infatti era troppo rigido: basarsi unicamente sulla ricerca di una coppia di curve che si sovrappongano alle discontinuità dell'immagine implica, quasi automaticamente, l'ipotesi di una strutturazione dell'ambiente spesso non presente. Per superare questa limitazione è stato proposto un approccio al problema che comprendesse sia elementi di elaborazione d'immagine (puntuali, locali e globali), sia elementi di ragionamento piú propri dell'intelligenza artificiale. Possiamo sintetizzare l'algoritmo proposto nel seguente modo:

- l'immagine viene *segmentata* sulla base di caratteristiche puntuali e locali del pixel, come varianza, colore, luminanza, tessitura; il risultato di questa elaborazione é un'insieme di gruppi di pixel contigui (cluster), omogenei per qualche insieme di proprietá.

- il passo successivo consiste nel individuare quale insieme di questi cluster puó rappresentare la strada con maggiore probabilitá; per fare questo si applica una tecnica chiamata *Decision Network* [10], che permette di valutare contemporaneamente due classi di fattori: 1) la probabilitá che un'area faccia parte della strada sulla base di caratteristiche intrinseche come forma, dimensione, tipo di terreno (valutato tramite la varianza dei suoi paramenti cromatici); 2) l'influenza che avrebbe la classificazione di tale area come strada sul comportamento del veicolo, tenendo conto della velocitá e direzione, con particolare attenzione al bilancio tra rischi e benefici di tali cambiamenti.

Si riconosce quindi la necessitá di un'elaborazione di alto livello, che tenga conto non solo del contenuto delle immagini elaborate, ma anche del complessivo stato del veicolo attuale, nel tentativo di minimizzare il rischio delle scelte fatte. Informazioni come velocitá, angolo di sterzo attuale, traiettoria prevista, prossimo way-point, devono necessariamente essere tenute in considerazione quando, per la strada, non esiste una codifica nota a priori e riconoscibile nelle immagini, come le linee dipinte

sull'asfalto. In questo caso l'informazione "dove vogliamo andare" ricopre un ruolo fondamentale nell'individuazione della carreggiata.

In Appendice B vengono illustrati i dettagli implementativi di questo sistema.

4.2 Rilevazione dei ostacoli in movimento agli incroci

Nel corso della Grand Challenge 2007 è stato sviluppato un sistema per l'individuazione del traffico in arrivo, utilizzato quando il proprio veicolo si trova fermo ad un incrocio. Caratteristica principale di tale sistema é la capacitá di rilevare oggetti in movimento a grande distanza, superiore ai 100m. L'algoritmo proposto si basa su una versione migliorata del *background subtraction* [11] [12], della quale gli aspetti piú interessanti sono:

- *rimozione delle vibrazioni*: essendo le telecamere installate su un veicolo con motore acceso, la immagini saranno affette da una vibrazione a carattere oscillatorio e periodico. Questo darebbe luogo ad una differenza diversa da zero nell'operazione di background subtraction, anche nelle zone dove non siano presenti oggetti in movimento. La soluzione proposta, in questo caso, é stata quella di effettuare una differenza tra l'immagine al tempo t e una somma pesata delle N precedenti: $t-1$, $t-2$, ..., $t-N$. Scegliendo opportunamente i pesi e N é stato possibile ottenere un'attenuazione quasi completa delle differenze dovute a vibrazioni, e anche di quelle dovute ad oggetti in movimento periodico, come, ad esempio, i rami degli alberi mossi dal vento;

- *rimozione del background*: soprattutto a grande distanza, lo spostamento di un oggetto tra un'immagine e quella all'instante successivo é spesso minimo; questo fa si che la differenza risulti diversa da zero non per tutta l'area ricoperta dal oggetto (veicolo), ma in genere solo per un sottile "bordo", anteriore e posteriore. Essendo la dimensione sull'immagine dell'intero oggetto giá molto piccola, questo effetto rischia di pregiudicarne completamente la rilevazione. Per superare il problema é stato proposto di rigenerare l'immagine di back-

ground come se gli oggetti in foreground non fossero presenti: si é stimato che nell'arco di 20/30 immagini un tipico veicolo si sposti per una lunghezza pari alla propria dimensione, dando quindi la possibilitá di osservare il background che prima occludeva con la sua presenza; é quindi possibile acquisire un'immagine, eliminare tutti gli oggetti in movimento, sostituendoli con le parti di background che, mano a mano, andranno a rendere visibile, ottenendo una nuova immagine di background che corrisponderà a quella che sarebbe visibile se nessun oggetto in movimento fosse mai stato presente sulla scena. Questa nuova immagine verrá utilizzata per applicare il background subtraction, accentuando notevolmente la visibilitá dei veicoli nell'immagine differenza.

In Appendice C vengono illustrati i dettagli implementativi di questo sistema.

Capitolo 5

Conclusioni

Scopo di questo lavoro di tesi era lo studio di un'architettura software specifica per veicoli autonomi. Sulla base dell'esperienza accumulata durante gli anni di Dottorato è stata proposta una soluzione di tipo distribuito, della quale l'elemento fondamentale è il *servizio*. Esso è definito come un processo che implementa una coppia di interfacce, le quali descrivono l'insieme delle procedure e dei messaggi che vengono utilizzati per comunicare ed interagire. Parallelamente è stata proposta una possibile suddivisione del sistema veicolo in una serie di specifici servizi, descrivendone gli scopi e le funzionalità.

La rappresentazione fornita è volutamente di alto livello, allo scopo di lasciare libertà agli sviluppatori di decidere i dettagli implementativi, e di non vincolare la struttura ad una particolare visione dei componenti che formano un veicolo autonomo. Nella codifica dei messaggi in formato XML si è invece sceso maggiormente nei dettagli: questo per mostrare al lettore un esempio concreto di traduzione dell'architettura in termini di programmazione reale, sia per mostrare i vantaggi nell'uso di questo linguaggio dal punto di vista della flessibilità e configurabilità. In realtà sarebbe semplice partire dagli esempi proposti e realizzare una nuova sintassi per uno specifico veicolo, semplicemente cambiando pochi campi.

Naturalmente esistono molti altri livelli in cui un veicolo autonomo può esse-

re descritto, pensiamo ad esempio al punto di vista hardware, con la topologia dei processori e dei sensori. O ancora alla struttura del sistema "x-by-wire", e tutti i collegamenti fisici con gli attuatori. Gli stessi elementi software possono a loro volta trovare una descrizione specifica nel linguaggio e nelle tecniche di programmazione utilizzate per implementarli. Trattarli tutti sarebbe impossibile. Quello che si è voluto portare con questa tesi è soprattutto un contributo di ordine e razionalizzazione agli elementi di alto livello, che sono certamente i meno consolidati e maturi, in quanto solo negli ultimi anni oggetto di studio approfondito.

L'architettura proposta deve certamente qualcosa alla disciplina dei *sistemi distribuiti*. La scomposizione del sistema in elementi di elaborazione come i servizi, connessi tramite rete e coordinati attraverso messaggi, può facilmente ricordare soluzioni simili proposte in altri ambiti, dove i sistemi distribuiti sono applicati da lungo tempo. La trattazione che ne abbiamo fatto non ha però la pretesa di inserirsi in modo formale e rigoroso in questo contesto, che è abbastanza lontano dall'ambito di ricerca che ha caratterizzato il mio Dottorato di Ricerca. È soprattutto il tentativo di trovare una sintesi, ed una riformulazione in termini formali e precisi, all'esperienza accumulata in questi anni, fatta di conoscenza ottenuta più per via empirica che per studio diretto. In particolare osservando come, nel settore dei veicoli autonomi, gli sforzi dei ricercatori siano stati rivolti soprattutto al risultato, allo studio degli specifici algoritmi (dalla pianificazione del moto alla sensorialità), trascurando la visione globale del sistema in cui questi elementi andavano ad inserirsi. Questo è in parte dovuto al modo in cui la ricerca in questo settore è stata ridestata, dopo anni di declino, passato attraverso delle competizioni, con premi monetari molto elevati, che hanno spostato l'attenzione più sulla prestazione finale che sullo studio formale di questi sistemi. Oggi, dopo alcuni anni dalla prima Grand Challenge, è forse iniziata una fase di revisione critica all'interno dei team che vi hanno partecipato, anche alla ricerca di una formalizzazione più razionale dei risultati ottenuti e di ciò che è stato prodotto, e della quale questa tesi vorrebbe essere un piccolo contributo.

Appendici

Appendice A

An Agent Based Evolutionary Approach to Path Detection for Off-road Vehicle Guidance

Abstract

This paper describes an ant colony optimization approach adopted to decide on road-borders to automatically guide a vehicle developed for the DARPA Grand Challenge 2004. Due to the complexity of off-road trails and different natural boundaries of the trails, lane markers detection schemes cannot work. Hence border detection is based on Ant Colony Optimization technique. Two borders at two sides of the road (as seen by a camera fixed on the vehicle) are tracked by two agent colonies: agents' moves are inspired by the behaviors of biological ants when trying to find the shortest path from nest to food. Reinforcement learning is done by pheromone updating based on some heuristic function and by changing the heuristic balancing parameters with the experience over the last tracked results. Shadow removal has also been introduced to increase robustness. Results on different off-road environments, as provided in the DARPA

Grand Challenge 2004, have been shown in the form of correct detections, false positives and false negatives and their dependence on number of ant-agents and balancing edge exploitation and pheromone-exploitation.

A.1 Introduction

The 13th March 2004, in the desert areas of the Mojave between California and Nevada, the first competition for fully autonomous ground vehicles took place: the DARPA Grand Challenge 2004 [1]. The challenge was to develop an autonomous vehicle able to complete a long and difficult off road path from Los Angeles to Las Vegas. The course could include unsurfaced roads, sandy and rocky trails, bushes, dry lakes and, in small percentage, paved roads. The course was supposed to be about 200 miles long, was defined by several thousand GPS way-points, and was not revealed until two hours before the event begun. The vehicles had to follow the GPS way-points as well as avoid natural and artificial obstacles, or other vehicles, without any human control. This paper presents an artificial vision algorithm developed as a part of the TarraMax vehicle. TerraMax is the results of Oshkosh Trucks Co., Ohio State University, and University of Parma partnership [13]. In particular the Artificial Vision and Intelligent Systems Lab of the University of Parma supplied a complete image processing system that, together with many other active sensors, performed the environment sensing: obstacle detection based on stereo vision, free space detection, and path detection based on monocular images, which is discussed this paper.

Lane detection on structured environments was already successfully faced using monochromatic monocular images by the authors [14] and other research teams [15], based on some *a priori* knowledge, like the existence of lane markers. Unfortunately this approach could not work in unknown environments. Figure A.1 shows some typical scenarios of the Grand Challenge path acquired by Terramax. Stones or bushes can be considered either road markers or can be even obstacles *inside* the road boundaries, by simply varying their position. Sometimes (Figure A.1(c)) it is just impossible, even for a human being, to localize any road.

(a) (b) (c)

Figura A.1: Possible roads as acquired by Terramax

The proposed method differs from traditional deterministic algorithm developed by other competing teams [16], and falls into the class of evolutionary algorithms [17]. Evolutionary methods are based on natural metaphor, and try to simulate the adaptive capabilities of the biological organism. A considerable application field is *optimization*, in which we mapped the computer vision problem of path detection. This paper is organized as follows: Section A.2 will briefly introduce the evolutionary approach on which path detection is based; Section A.3 will describe low-level image processing, while Section A.4 will present the details of the evolutionary system. Section A.5 will present some experimental results together with a quantitative analysis of the algorithm performance.

A.2 Evolutionary approach

The underlining idea of this work is the following: each boundary of the road is just one particular curve on the image, starting from the bottom and finishing on the horizon line. If we could find some *local* properties describing the probability that a pixel belong to the boundaries of the roads, then it would be possible to formalize the problem as an optimization problem. The optimal solutions are the ones that overlap with the largest number of pixels belonging to the boundaries of the road. Clearly the cost of a solution is inversely proportional to the number of overlapping pixels.

The algorithm implemented on Terramax is based on the well known Ant Colony

Optimization(ACO) [8] [9]: a distributed meta-heuristic for combinatorial optimization problem, inspired by the communication system of the biological ants. Experimental observations [18] show that a colony of real ants, after a transitory phase, always find the shortest path from the nest to the food. This is achieved by an odorous substance, called pheromone, deposed by ants on the terrain, which attracts the following ants. Once arrived at a decision point, ants make a probabilistic choice based on the amount of pheromone they smell in correspondence to all the possible roads. Due to this *autocatalytich* process, the shortest path (the one covered in less time) will emerge as the one with the largest amount of pheromone. Therefore, returning to the vision problem, the goal became to build two colonies –one for each side of the road (left and right)– made of agents that, moving pixel by pixel, try to find the optimal boundary curve on their side.

Ant Colony Optimization was already successfully applied to solve significant combinatorial optimization problem, as the Traveling Salesman Problem [19], distributed network routing [20], vehicle routing [21], but it is not very commonly used in computer vision; in particular there is no known application to autonomous vehicle guidance problems. The algorithm described in this paper is the only evolutionary approach to path detection seen during the DARPA Grand Challenge.

A.2.1 The ACO approach

A generic representation of a combinatorial optimization problems which can be solved by artificial ACO algorithms can be formalized as follows:

- A finite set X of *components* χ_i.
- A finite set Λ of *connections* λ_{ij} among components χ_i and χ_j in X.
- A *cost function* $\Gamma(\lambda_{ij},t)$ with which assign to each λ_{ij} its own cost value, defined as $\Gamma_{\lambda_{ij}} \equiv \Gamma(\lambda_{ij},t)$, where t is a time measure.
- A finite set $\Omega(X,\Lambda,t)$ of *constrains* between X and Λ elements.

- A set Σ of *states* defined as variable length sequences of components: $\sigma =< \chi_i, \chi_j, ..., \chi_k, ... >$
- A set $\widehat{\Sigma(t)} \in \Sigma$ if *feasible states*, defined by $\Omega(X, \Lambda, t)$
- A state σ_1 is a *feasible neighbor* of the state σ_2 if : (i) both σ_1 and σ_2 are in $\widehat{\Sigma(t)}$; (ii) σ_1 can be reached from σ_2 in one single step: since χ_i is the last element of the σ_2 sequence, σ_1 is a neighbor of σ_2 if $\exists \chi_j \in X \ : \ \exists \lambda_{ij} \in \Lambda$ and $\sigma_1 =< \sigma_2, \chi_j >$. The set of feasible neighbor of σ is called $N_\sigma(t)$.
- A *solution* Ψ is an element of $\widehat{\Sigma(t)}$ that meets some *exit condition e*
- An *objective cost function* Γ_Ψ which assigns a cost value to each solution reached

In other words the combinatorial optimization problem must be formalized in terms of shortest path problem through a graph $G =< X, \Lambda >$, where each solution is represented by a sequence of components χ_i (path) satisfying the constrains $\Omega(X, \Lambda, t)$ and meeting some *exit condition e*. The ACO algorithms are implemented by building a colony of artificial ants (agents), that tries to approximate the optimal solution in the following way:

- Each agent has one *starting state* σ_s^k and one, or more, *exit condition* e^k.
- Each agent starts from its own starting state and moves through one of its own feasible neighbors, thus building a solution step by step. The processing ends when the agent reaches its own exit condition.
- Often a *heuristic function* η_{ji} is supplied, to indicate the convenience of moving from a state σ_j to $\sigma_i \in N_{\sigma_j}(t)$ based on a priori knowledge.
- Each connection λ_{ji} has an associated *pheromone trail* $\tau_{\lambda_{ji}}$[1] encoding the experience of the previous agents: the higher the pheromone deposit $\tau_{\lambda_{ji}}$, the lower the costs of previous agents' solutions that included connection λ_{ji}.

[1] Or simply τ_{ji}

- Each agent on state σ_j moves towards one of its own feasible neighbors σ_i, through connection λ_{ji}, applying a *probabilistic rule*. The probabilistic rule is a function of: (i) the heuristic function η_{ji}; (ii) the pheromone trail $\tau_{\lambda_{ji}}$; (iii) the information yield by the agent during its past travel to the current state.

- When leaving a state σ_j towards state σ_i via connection λ_{ji}, each agent may update the pheromone trail $\tau_{\lambda_{ji}}$.

- Once an agent has reached the exit condition it increases the quantity of pheromone on its path in a way inversely proportional to the cost of the solution it constructed. This is a *reinforcement learning* strategy [22] that modifies the way the following agents will perceive the problem, thus performing a distributed adaptive approach.

- Since in this paper the ACO algorithm is assumed convergent, when all agents has reached a solutions the pheromone trails are analyzed to find the best path that approximates the optimal solution of the combinatorial optimization problem. The convergence to the optimal solution of the ACO algorithms has been studied in several papers [23] [24].

The next section shows how the path detection problem was mapped onto the Ant Colony Optimization formalism.

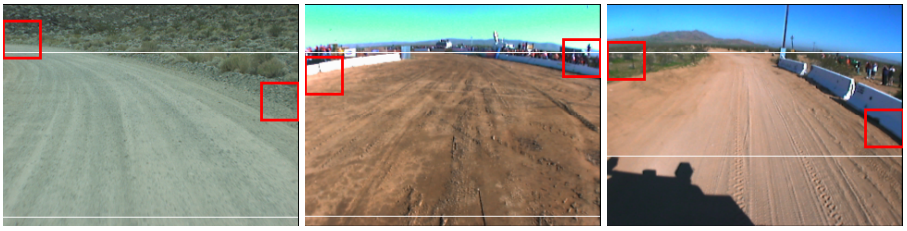
A.3 Preprocessing

Before applying the ACO algorithm, we need to develop low level preprocessing procedures to localize the optimal starting and final states and the appropriate local proprieties of road boundaries pixels.

A.3.1 Interest area: starting states, final states and set of components X

The final states and the set of components X are defined by the constrains of the output map. The vision sensor provides to the path planner a 50m x 50m map of

the space in front the vehicle, so 50m represents the upper limit for the agents' movement. Each component $\chi_i \in X$ is a pixel between the upper limit and the bottom of the image. Consequently a feasible state σ is any ordered pixel sequence starting form the bottom of the image; a final state Ψ is any state ending on the upper limit.



(a) (b) (c)

Figura A.2: Examples of different scenarios: the agents' starting areas are indicated by squares, the white lines indicate the area of interest

The initial states are the pixels inside the two starting areas shown in Figure A.2 as squares. They are 40x40 pixels squares, placed in peripheries areas where a sufficient percentage of edges is present. In very sharp curves it is possible that only one boundary of the road is visible: in this case it is very helpful to move the starting areas: as shown in Figures A.2(a) the start and final states overlap, and only one bound will be detected. Therefore, before starting the execution, each agent of the colony is put randomly on a pixel in these areas.

As shown in Figure A.2(c) a vehicle shadow removal step is also implemented to reduce the processing area. Basically this procedure is based on the idea that the *vehicle* shadow is always placed on the bottom of the acquired images. Hence the image is analyzed to localize a region *connected to the bottom* of the image, whose brightness is significantly lower (in percentage) than the average brightness of the whole image. If such a region exists, it is used to define the lower limit of the area of interest. The shadow removal step is necessary since the shadow could generate strong edges and could lead to false values of the road chromatic proprieties.

A.3.2 Heuristic and Cost function

An extremely important preprocessing step is the definition of the local properties that characterize the road boundaries in unknown and unstructured environments. To do this a special image representing distance to the average of the road, called *imD*, is built from the 320x240 acquired RGB image. The assumptions are: the vehicle is already on the road and the road is homogeneous.

The *imD* image is computed as follows: the 640x480 pixels image in the RGB color space is converted to a 320x240 pixels in a brightness independent domain: the RGB Normalized (RGBN). Then a temporal average of Red, Green and Blue values of road pixels is computed, and called $\overrightarrow{E_{road}(t)}$: this depends on all the images acquired so far. The application of *vectorial euclidean distance function* $d(\overrightarrow{X}, \overrightarrow{Y})$ leads to a monochromatic image *imD*. Figure A.3 shows an example of RGBN image (a) and the corresponding distance to the average (b). Finally, applying a gradient operator, we obtain a monochromatic edge image (Figure A.3(c)), that represents the only a priori information given to the agents. The brighter a pixel, the higher the probability that it belongs to the boundaries of the road. This is due to the nice segmentation achieved in *imD* (Figure A.3(b)) that enhances the differences between pixels inside and outside the road, and, at the same time, smooths out the differences within those two classes. So, the attractiveness of a pixel is proportional to the brightness of its correspondent pixel in the edge image (this rule defines the *local heuristic function*) [25]. In the same way, the cost of the movement towards a pixel is inversely proportional to the brightness of its correspondent in the edge image (this rule defines the *cost function*).

Figure A.4 summarizes the whole block diagram of the preprocessing, where all the information needed by the ACO algorithm are depicted.

(a) (b) (c)

Figura A.3: RGB (a) and distance to the average image (b). A gradient operator computed to obtain the edge image (c)

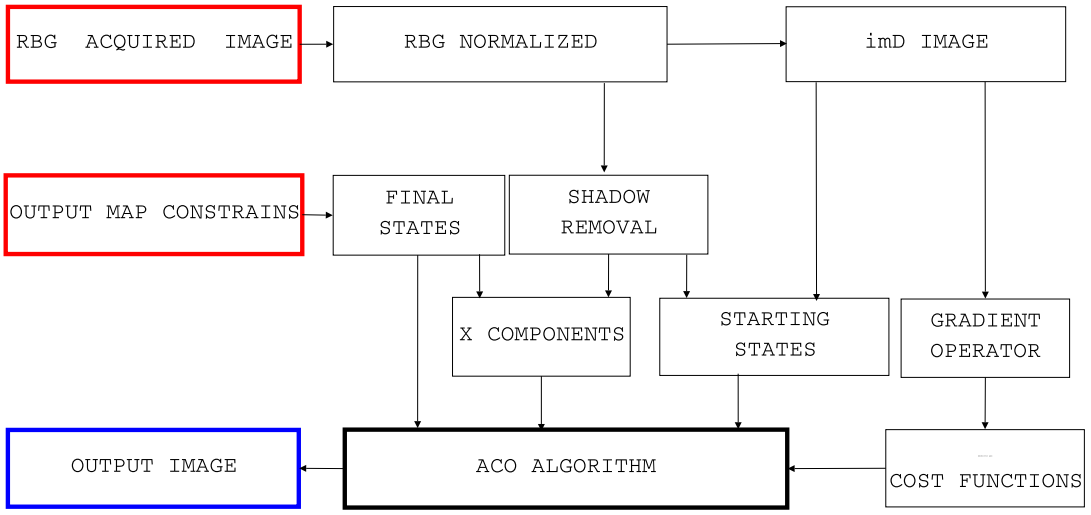


Figura A.4: Preprocessing block diagram

A.4 Application of the ACO algorithm

A.4.1 The point of attraction

Before defining the moving rules, it is necessary to define the set of *feasible neighbors* $N_{\sigma h}$ of a state σ_h. Given a pixel (i_a, j_a):

- an agent on $\sigma_h =< \sigma_{h-1}, (i_a, j_a) >$ can move only towards the set of pixels of

coordinates $\{(Center(P,i_a,j_a)-3,j_a),...,(Center(P,i_a,j_a)+3,j_a)\}$;

- in case of *backtracking* (see Section A.4.2) the agents can move towards pixels $\{(Center(P,i_a,j_a)-3,j_a-1),...,(Center(P,i_a,j_a)+3,j_a-1)\}$;

where (i_a,j_a) are the row-column coordinates of the current pixel, and $Center(P,i_a,j_a)$ is a function of the current pixel and the *point of attraction* $P=(i_p,j_p)$. Assuming an orthogonal reference system in which the point of coordinates $(0,0)$ is placed in the upper-left image corner, as shown in Figure A.5, the function $Center(P,i_a,j_a)$ is defined as follows:

$$Center(P,i_a,j_a) = \text{the nearest integer to } r(j_a-1) \tag{A.1}$$

where $r(j)$ is the line connecting the point of attraction $P=(i_p,j_p)$ and the current position of agent (i_a,j_a) (see Figure A.5). The function $r(j)$ is computed as follows:

$$r(j) = i_p + \frac{i_a-i_p}{j_a-j_p}\cdot(j-j_p) \qquad j_a \neq j_p \text{ is always true by definition} \tag{A.2}$$

The point of attraction is computed frame by frame, using information on previously obtained road boundaries, as shown in Figure A.6.

The point of attraction polarizes the random moving component of the agents. Therefore, if the agents chose the next pixel in a *purely* random way, then the average moving direction of the colony is towards the point of attraction (Figure A.7). The possibility of changing the point of attraction is of basic importance since the agents has limited capabilities of exploration, due to the limited set of feasible neighbors. Therefore it is necessary to provide some help to the agents in their searching movement, giving a preferential direction.

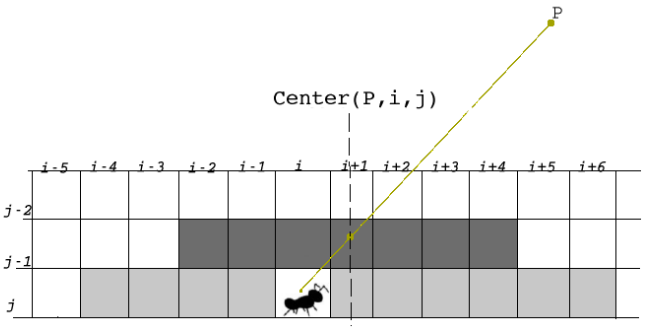
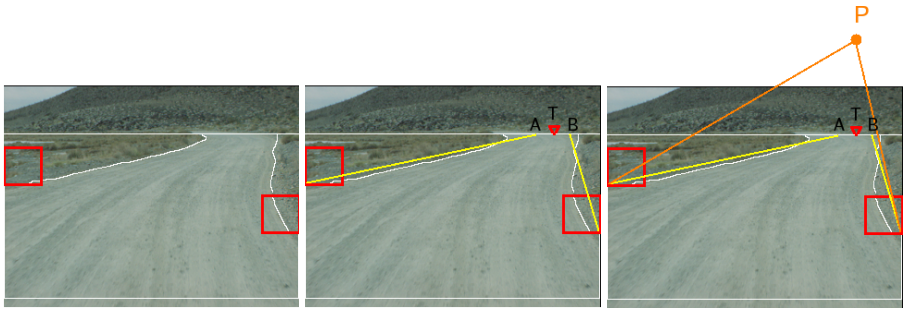


Figura A.5: The set of neighbors of (i, j) pixel: in light gray the neighbors in case of backtracking movement, in dark gray the neighbors in case of forward movement



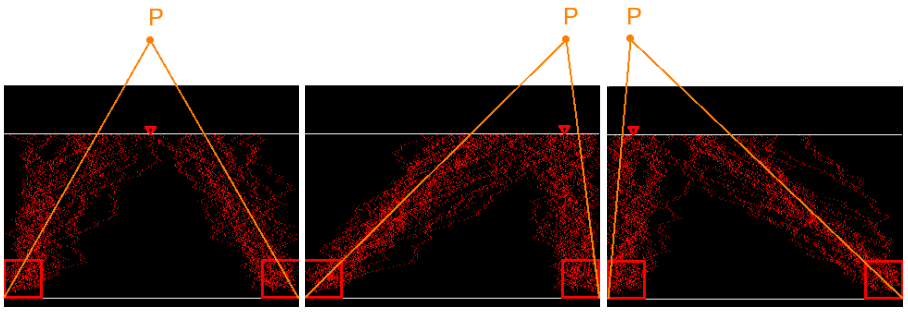
(a) (b) (c)

Figura A.6: Example of how the point of attraction is computed. The white curves on (a) represent the road boundaries obtained at time t, the straight lines in (b) represent their linear approximation and the triangle T is the midpoint between A and B. Thus the point of attraction at time $t+1$ will have the same coordinate i_p of T at time t, while coordinate j_p is fixed for all possible points of attraction at any time.

A.4.2 Motion rules

The motion rules can be divided into three levels. The first one is the well known *random-proportional* [9] rule for artificial ants[2]:

[2]Hereinafter we will simplify the notation and indicate $\chi_k \in N^*_{\sigma_h}$ as $\chi_k \in X : <\sigma_h, \chi_k> \in N^*_{\sigma_h}$, where $N^*_{\sigma_h}$ is any subset of N_{σ_h}.



(a) (b) (c)

Figura A.7: Examples of the effect of different points of attraction P: the highlighted pixels represent the trails covered by the agents of each colony when moving randomly

$$a_{\chi_k} = \begin{cases} \dfrac{(\alpha)\cdot\tau_{hk}(t)+(1-\alpha)\cdot\eta_{hk}}{\sum_{\chi_l\in\overline{N_{\sigma_h}}}(\alpha)\cdot\tau_{hl}(t)+(1-\alpha)\cdot\eta_{hl}} & \text{if } \chi_k \in \overline{N_{\sigma_h}} \\ 0 & \text{otherwise} \end{cases} \tag{A.3}$$

where:

- σ_h is the current state.
- a_{χ_k} is the probability of state $< \sigma_h, \chi_k >$ of being the next state.
- $\overline{N_{\sigma_h}} = N_{\sigma_h} \cap \{(r, j-1), r \in N\}$ are the neighbors on row $j-1$.
- α is a parameter with which it is possible to tune the balance between edge-exploitation and pheromone-exploitation behavior of the agents.

The second level consists of the so-called *pseudo-random-proportional* rule, with which it is possible to improve the exploitation behavior of the agents:

$$\chi_{next} = \begin{cases} \chi_j : \eta_{\chi_k} = max_{\chi_l\in\overline{N_{\sigma_h}}}(\eta_{\chi_l}) & \text{if } q \le q_0 \\ \chi_k \text{ random based on Formula (A.3)} & \text{otherwise} \end{cases} \tag{A.4}$$

where q is a random variable with a uniform probability distribution, and q_0 is a threshold defined as follows:

$$q_0 = \gamma \cdot \frac{max_{\chi_l \in \overline{N_{\sigma_h}}}(\eta_{\chi_l} - \eta_{\chi_{next}})}{max_{\chi_l \in \overline{N_{\sigma_h}}}(\eta_{\chi_l})} \tag{A.5}$$

Parameter γ is used by the programmer to vary the balance between global *exploitation* and *exploration* behavior. If the sum of pheromone trails and heuristic values of $\overline{N_{\sigma_h}}$ is zero, agents consider this movements too profitless, and perform backtracking steps. During backtracking agents do not proceed towards pixels on the upper row $(j-1)$, but towards pixels on the current row in $\widetilde{N_{\sigma_h}} = N_{\sigma_h} \cap \{(r, j), r \in N\}$. Each pixel $\chi_k \in \widetilde{N_{\sigma_h}}$ is chosen with a probability:

$$p_{\chi_k} = \begin{cases} \dfrac{\eta_{\chi_k}}{\sum\limits_{\chi_l \in \widetilde{N_{\sigma_h}}} \eta_{\chi_l}} & \text{if } (\chi_k \in \widetilde{N_{\sigma_h}}) \wedge (\chi_k \neq \chi_h) \\ 0 & \text{otherwise} \end{cases} \tag{A.6}$$

This procedure could lead to a deadlock situation, so backtracking is permitted only once for each row. After this, a purely random move is generated. The main goal of this procedure is to speed up the convergence of the algorithm: even without backtracking, agents will be able to skip the regions with no pheromone and edges, but backtracking accelerates this process, allowing to decrease the number of agents needed to achieve the convergence.

A.4.3 Pheromone update

The N_a agents of a colony are divided into different subsets called $N_{a1}, N_{a2}, ..., N_{an}$, each one characterized by different moving rules parameters, and executed in sequence. When every agent of a particular subset has reached the final pixels, the pheromone trails $\tau(t)$ are updated as follows:

$$\tau_{ji}(t+1) = (1-\rho) \cdot \tau_{(ji)}(t) + \rho \cdot \sum_{k=1}^{N_{ax}} \Delta_{ij}^{k}(t) \tag{A.7}$$

where:

- t is a time measure, and represents the evolution of the pheromone deposit.
- $\rho \in (0,1]$ is the pheromone *evaporation ratio*.
- Δ_{ij}^{k} is the k-ant contribution, computed as follows:

$$\Delta_{ij}^{k} = \begin{cases} Q/(L_k - L_{k-best}) & \text{if k-ant passed through } \lambda_{ij} \\ 0 & \text{otherwise} \end{cases} \tag{A.8}$$

where Q is a fixed parameter. L_k is the cost of the k-agent, and L_{k-best} is the cost of the current best solution respectively, computed as follow:

$$L_k = \frac{\sum_{ij \in path_k}(255 - edge_{ij})}{\text{path length}} \tag{A.9}$$

so the path cost is inversely proportional to the average brightness accumulated over the collection of its pixels: a path formed by bright pixels costs less than a path formed by dark pixels.

A.4.4 Management of agents

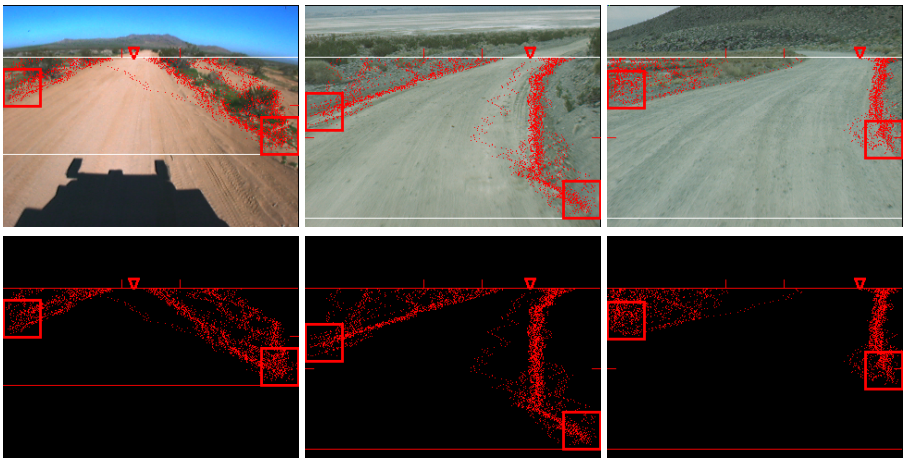
As anticipated, each colony has N_a agents, divided into several subsets N_{a1}, N_{a2}, ..., N_{an}, each one characterized by different rules parameters:

- Agents in N_{a1} have $\alpha = 0$. From the analysis of Formulas (A.3) and (A.4) it is clear that this means that the agents movement is only based on heuristics, ignoring the pheromone deposit. This could be considered as an *edge-exploitation* phase, in which previous experience is not considered.
- The agents in subset $N_{a2}, ..., N_{an}$ have parameter α defined as follow:

$$\alpha_i = \frac{i}{\text{Number of subset}} \cdot \alpha_p \tag{A.10}$$

where α_p is a value defined by the programmer, constant for all executions. In this way, as the execution proceeds, agents become more and more sensitive to pheromone, and less to heuristics (*pheromone-exploitation*). The underlying ratio is that the pheromone deposit becomes more and more reliable as time goes by, because it represents the experience of a large number of agents.

Figure A.8 shows the agents' paths, represented by a sequence of highlighted pixels. Note that the paths are more concentrated on the actual road boundaries.



(a) (b) (c)

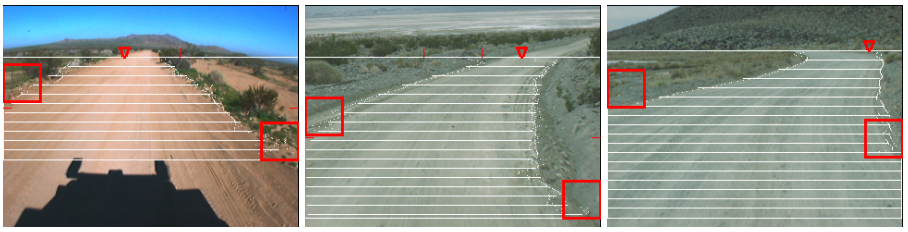
Figura A.8: The agents' paths: the upper row presents the original images with the agents' paths superimposed, the bottom row shows only the ants' paths for an easier visual evaluation

A.4.5 Solution extraction

To achieve the final solution two special agents, one for each colony, are created: they start from the starting areas and move pixel by pixel until a final pixel is reached. They are attracted *only* by the pheromone trails, and ignore the heuristic function.

The two solutions obtained in this way, one for each colony (left and right), will represent the final road boundaries. The two sequences of white dots shown in Figu-

re A.9 represent the special agents' paths; the space between them is considered as the surface of the road and represented by white lines.



(a) (b) (c)

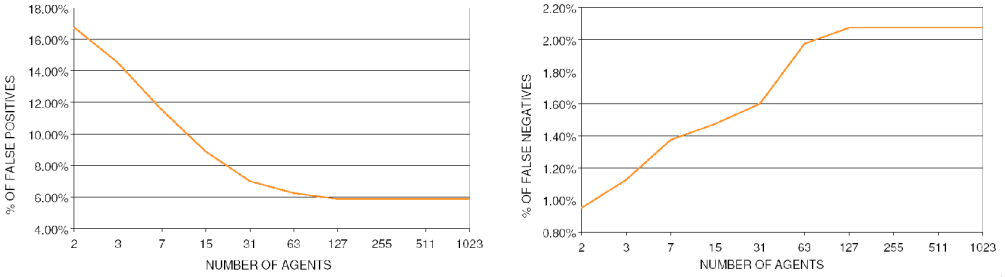
Figura A.9: Some results of correct path detection: the path is indicated by white horizontal lines. The vehicle shadow removal reduced the processing area in figure (a).

A.5 Experimental results

There are many parameters that influence the algorithm behavior: the number of agents, the balance between exploitation and exploration, pheromone upgrade parameters, pheromone evaporation ratio, a few thresholds, etc. Some of these parameters has been chosen empirically to achieve the best results: pheromone evaporation ratio $\rho = 0.1$ in Formula (A.7); pheromone update contribution $Q = 1$ in Formula (A.8); in the shadow removal procedure, a region of pixels is considered as shadow when its brightness is $\leq 45\%$ of the average image brightness.

Some others parameters deserve a much more detailed discussion, since they are closely related to the multi-agent distributed nature of the proposed system. The first one, the number of agent for each colony, has also a great influence on the global performance. According to the order in which agents are executed (see in Section A.4.4), we made some systematic tests, using presegmentated images[3]. The human and algorithm's results were compared, measuring the error percentage. Figure A.10 shows

[3]There images were manually marked by a human operator that labeled road and off-road areas



(a) (b)

Figura A.10: Tests on the number of agents: false positives (a) and false negatives (b) percentage. The average values were computed for 100 runs of the algorithm. The subdivision of the subsets is made in the following way: given $N_a = 2^n - 1$, the subsets size has to be: $N_{1a} = N_a/2$, $N_{2a} = N_{1a}/2$, $N_{3a} = N_{2a}/2$, ..., $N_{na} = 1$.

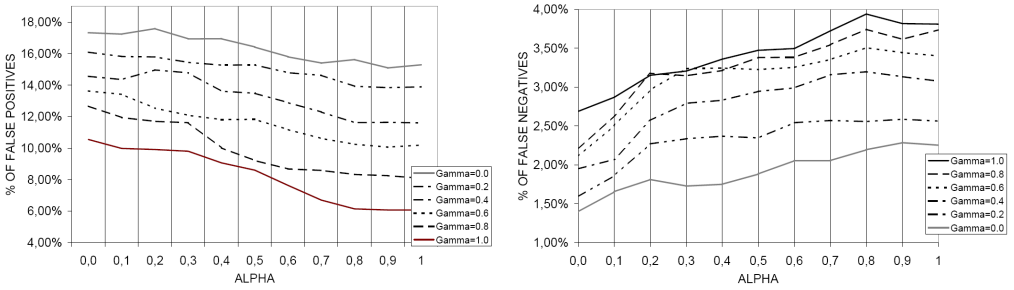
the error percentage as a function of the number of agents: *false positives* pixels are here defined as the ones classified as road by the algorithm but as off-road by the human operator, while *false negatives* are the pixels classified as off-road by the algorithm but as road by the human operator.

Number of agents	2	3	7	15	31	63	127	255	511	1023
Time (ms)	48	60	66	73	80	87	97	110	131	164

Figura A.11: Average execution time. Hardware: Intel Pentium IV 3GHz, 1GB RAM, Linux OS. Acquisition frame rate: 10 per second.

This test shows an interesting behavior of agents: when using a small number of agents the majority of the solutions is correctly attracted by areas with a high density of edges, but the results include many false positives and just a few false negatives. When raising the agents number, the agents tent to cover paths closer and closer to the road shape. Considering also the algorithm's time performance on the TerraMax hardware (shown in Figure A.11), the optimal number of agents is defined to be 63 for each colony.

Two more parameters that were tested are $\alpha_p \in [0,1]$ and $\gamma \in [0,1]$ defined in



(a) (b)

Figura A.12: Tests on the number of agents: false positives (a) and false negatives (b) percentage. These test were performed under the same conditions described in the Figure A.10.

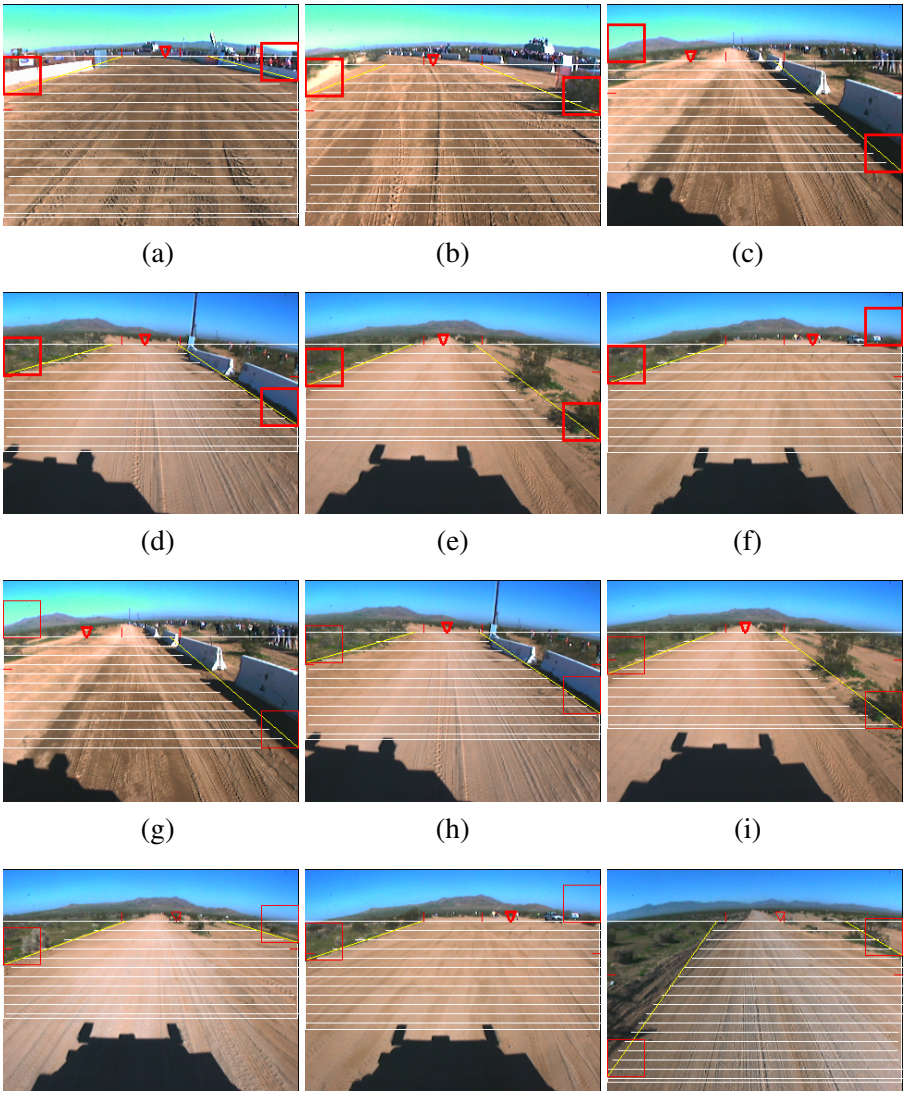
Formula (A.3) and (A.4). The higher is α, the more agents are attracted by edges, and the lesser are attracted by pheromone; the higher is γ, the more agents have an edge-exploitation behavior, and the lesser have an exploratory behavior. Figure A.12 shows an interesting result: the error percentage is almost independent of α_p, while on the other hand it is highly dependent on γ. This is easily explained since α is ruled by the agents execution cycle (seen in Paragraph A.4.4): that mechanism prevents the disastrous effect obtained when all agents use a static $\alpha = 1$, that leads the agents to ignore the heuristic at all. The best results have been achieved with $\alpha_p \geq 0.8$. The great influence of γ shows the importance of the heuristic function, that represents the only knowledge of the agents about the problem. This can be considered as a proof of the highly significant preprocessing, since the rule in Formula (A.4) becomes more performing as it becomes more dependent on the heuristic function.

A.6 Conclusion

This paper presented an application of Ant Colony Optimization to the road-borders detection problem, specifically developed for the DARPA Grand Challenge 2004 and tested on the Terramax autonomous vehicle. The ACO approach is well suited for

a set of applications (for example the distributed network routing, or the Traveler Salesman Problem), but its application to solve the path detection problem was not straightforward. Therefore the basic ACO algorithm, well known from the literature, has been improved with some new features: polarization of the random movement by the point of attraction (Paragraph A.4.1), the mechanism which modifies the alpha parameter during the processing of each frame (Paragraph A.4.4), the ability of changing the starting states of agents frame by frame (Paragraph A.3.1). These improvements complemented the ACO algorithm and made it suitable for the path detection problem.

The Artificial Vision and Intelligent Systems Lab of the University of Parma developed, for the DARPA Grand Challenge 2004, two more algorithms for path detection: one based on 3D models, one based on region growing using color and texture information.All these algorithms were tested on images taken during the DARPA Grand Challenge, and the ACO approach demonstrated to be robust in many critical situations. The main weakness which is still present in the proposed algorithm is the inability to detecting more than one pair of road-borders, which typically happens when approaching crossroads or junctions. The next step, looking to the DARPA Grand Challenge 2005, will be allowing more than 2 agent colonies to detect all the road-borders candidates, and thus implementing a higher level processing to recognize what kind of path the vehicle is actually facing (single road, crossroads, multiple lane road, etc). Figure A.13 shows some real results.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(l) (m) (n)

Figura A.13: Some results form the Grand Challenge 2004: the Grand Challenge starting area (a)(b); the first curve (c); the first long straight road where some artificial boundaries are visible(c)(d); a long straight road delimited by vegetation (e)(f).

Appendice B

A Decision Network Based Frame-work for Visual Off-Road Path Detection Problem

Abstract

This paper describes a Decision Network based frame-work used for path detection algorithm development in autonomous vehicle applications. Lane marker detection algorithms do not work in off-road environments. Off-road trails have too much complexity, with widely varying textures and many differing natural boundaries. The authors have developed a general approach. Images are segmented into regions, based on the homogeneity of some pixel properties and the resulting regions are classified as road or not-road by a Decision Network Process. Combinations of contiguous clusters form the path surface, allowing any arbitrary path to be represented.

B.1 Introduction

On October 8th, 2005 twenty-three vehicles and no drivers gathered in the Mojave desert to compete in the second edition of the DARPA Grand Challenge [2]. The US Defense Advanced Research Programs Administration (DARPA) created this robotic vehicle competition as an open challenge intended to energize the engineering community to tackle the major issues confronting autonomous vehicle development. For the timed competition, DARPA designed a 132-mile off-road desert course that each vehicle had to negotiate. The course was defined by an ordered list of geographic waypoints, a maximum speed for each waypoint and boundaries that could not be crossed. Vehicles had to operate with full autonomy as they maneuvered around obstacles lining the desert course.

This paper presents an artificial vision algorithm developed as a part of the TerraMax vehicle. Oshkosh Truck Corporation, Rockwell Collins and the University of Parma partnered together to form Team TerraMax™[26]. The Team TerraMax™ robotic vehicle is a US Marine Medium Tactical Vehicle Replacement (MTVR) truck which was fitted with electronic actuators for steering, brake, throttle, and transmission control. A computer network was installed to host the software applications necessary for autonomous navigation. The applications consisted of vehicle control, path planning, LIDAR obstacle detection, and artificial vision obstacle detection and path following.

The Artificial Vision and Intelligent Systems Lab (VisLab) of the University of Parma developed the artificial vision systems that sensed the environment. Three color cameras captured video images; a single computer processed the data. Obstacle detection used stereo vision and a v-disparity approach. Path detection used monocular images and the approach discussed in this paper.

B.2 Path Detection as a Decision Problem

Path detection on structured environments has been already successfully faced by the authors [14] and others [27], using monochromatic monocular images and assuming the existence of lane markers. Unfortunately on unstructured and unknown environments is not possible to rely on any *a priori* knowledge about road structure. To overcome this lack of information many different approaches have been proposed: learning the road properties by neural networks [28], selecting the actual road-borders within the set of all possible curves on the image by evolutionary techniques [29], and growing regions believed to belong the road, on the basis of some a priori assumptions or simplifications. All these methods look for a single homogeneous road surface in front of the vehicle, searching based on brightness, color, texture, etc. The hypothesis of uniform homogeneity becomes a huge limitation as it bounds the set of detectable roads to the case of medium/well structured environments.

The proposed method differs from the above methods, overcoming their limitations by generalizing the problem. Roads can have *heterogeneous* surfaces. To find these potentially heterogeneous surfaces, the algorithm looks to build them up from a variable number of smaller *homogeneous* terrain portions. The homogeneous portion of terrain can represent any kind of natural or artificial environment elements, such as gravel or paved roads but also grass, water paddle, oil stains, drivable rocks, lane markers, shadows, etc, and they do not need to be previously learned and recorded in a database [30]. Consequently it is possible to summarize the path detection algorithm as a two-step process:

- divide the image in homogeneous regions made of connected pixels.
- decide which combination of the obtained regions could represent the road surface with the highest probability.

The first step is called clustering, or image segmentation. Clustering in computer vision has been being studied with success, especially for medical applications, using both evolutionary [31] and traditional [32] [33] approaches. However the real-time

constraints of the Grand Challenge contest led to the adoption of a simple but fast and easily tunable clustering algorithm, explained in Section B.3, as a good trade-off between performance and computational requirements.

The second step falls in the class of *decision problems*. Born to help decision processes in medical, transportation, political or environmental fields, *decision theory* [34] is now widely used by Artificial Intelligence researchers as a useful frame-work into which they can map a variety of classical problems. Decision Networks [10] extend Bayesian networks, adding actions and utilities to provide a general methodology for rational decision making. Section B.4 will describe how Decision Network fits the problem of decide about the set of clusters that belong to the road surface. The decision process tries to minimize the risk of wrong classifications taking into account the current *vehicle state* given by speed, steering angle, steering angle acceleration.

This approach has the advantage of explicitly shifting the path detection to a high level problem, allowing a wider range of situations to be handled in a more sensible way. For example there is no need to remove vehicle shadows at a medium/lower level, on the basis of brightness considerations. In fact since vehicle shadows belong to the terrain surface, its corresponding cluster will be treated just as any other region: like a potential part of the road.

B.3 Clustering

Several image segmentation algorithms can be found in literature. The most common approach is to join close pixels to obtain a large region composed by similar entities. In approach described in this section, images are decomposed in $d \cdot d$ pixels cells and a comparison is made among them. The use of cells instead of pixels allows a comparison using both the average color value and the information about the texture, like variance.

B.3.1 Pseudo Distance Function

To measure the similarity of cells we defined a *pseudo distance function*, that combines distances from cell to cell, from cluster to cluster, and from cell to cluster. Before introducing the distance function it is necessary to define the following entities:

- A set C of $c_{i,j}$, where $c_{i,j} =< x_{k1}, x_{k2}, ...x_{kn} >= c_k \in \Re^n$ is the properties vector of the $k-th$ *cell*, placed in position i, j on the interest area of the image. The interest area is made of the cells that correspond to position not farther that 50m from the vehicle. This information is obtained exploiting the methods described in [35] (See Fig. B.2).

- A set V of v_k, where $v_k =< y_{k1}, y_{k2}, ...y_{kn} >\in \Re^n$ is the properties vector of the $k-th$ *cluster* of cells [1].

The partial comparison functions are defined as follows:

- the *cell to cell only* comparison function is:

$$c2c(c_k, c_l) = \sum_{i=0}^{n} D_{i-c2c}(x_{ki}, x_{li}) \cdot \alpha_i, \forall c_k, c_l \in C \tag{B.1}$$

- the *cluster to cluster only* comparison function is:

$$v2v(v_k, v_l) = \sum_{i=0}^{n} D_{i-v2v}(y_{ki}, y_{li}) \cdot \alpha_i, \forall v_k, v_l \in V \tag{B.2}$$

- the *cell to cluster only* comparison function is:

$$c2v(c_k, v_l) = \sum_{i=0}^{n} D_{i-c2v}(x_{ki}, y_{li}) \cdot \alpha_i, \forall c_k \in C \text{ and } v_l \in V \tag{B.3}$$

[1] Hereinafter we will use v_k meaning both the set of cells contained in the $k-th$ cluster and the corresponding properties vector

where $\alpha_i \in \Re^+$ is the fixed weight assigned to the $i-th$ cluster and cell properties, and $D_i(\cdot,\cdot)$ is the corresponding *property comparison function*.

The properties comparison functions must always be greater or equal to zero, but no other constraints are required by our frame-work.

Finally we can define the overall cell to cell *pseudo distance function* $Dist(\cdot,\cdot)$ as follows:

- If c_k,c_l belong, respectively, to the clusters v_k,v_l:

$$Dist(c_k,c_l) = c2c(c_k,c_l)\cdot(1-\beta_c) + v2v(v_k,v_l)\cdot\beta_c \tag{B.4}$$

- If only c_l belongs to a clusters (v_l):

$$Dist(c_k,c_l) = c2c(c_k,c_l)\cdot(1-\beta_c) + c2v(c_k,v_l)\cdot\beta_c \tag{B.5}$$

- If only c_k belongs to a clusters (v_k):

$$Dist(c_k,c_l) = c2c(c_k,c_l)\cdot(1-\beta_c) + c2v(c_l,v_k)\cdot\beta_c \tag{B.6}$$

where $\beta_c \in \Re^+$ is the fixed weight given to the clusters distance when computing the distance between a pair of cells.

Two cells c_k, c_l are *similar* if and only if:

$$Dist(c_k,c_l) \leq q_0 \tag{B.7}$$

where q_0 is a fixed threshold. Note that $Dist(\cdot,\cdot) \geq 0$, but in general we cannot say nothing about $Dist(a,b) = 0$ and the commutative property. The function $Dist(\cdot,\cdot)$ is applied on cells, but it also takes into account the cluster to which the cells belong, hence the distance two cells with the same characteristics vector could have a distance greater than zero, depending on their clusters. For this reason cells

should be selected for comparison based on a *pseudo-random* method, as explained in the next Paragraph.

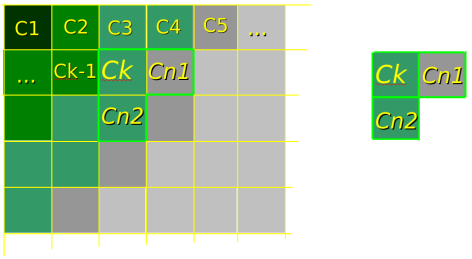


Figura B.1: The 2-neighbors cells c_{n1}, c_{n2}

B.3.2 Clustering algorithm

For every pseudo-randomly chosen $c_k \in C$, the clustering process involves the following steps:

1. If c_k does not belong to any cluster a new cluster $v_k = c_k$ is created. This cluster actually corresponds to the c_k cell only.

2. The distances D_{n1}, D_{n2} to the 2-neighbors (see Fig. B.1) cells c_{n1}, c_{n2} are computed by (B.7). Suppose that $D_{n1} < D_{n2}$.

3. When c_{n1} belong to a cluster $v_{n1} \in V$ the following rules, in order of priority, apply:

   - $v2v(v_k, v_{n1}) < q_{c0} \;\rightarrow$ the clusters v_k and v_{n1} will be merged, averaging the vectors properties. q_{c0} is a fixed threshold.
   - $(c2v(c_k, v_{n1}) < c2v(c_k, v_k)) \wedge \neg(c2v(c_{n1}, v_k) < c2v(c_{n1}, v_{n1})) \;\rightarrow$ the cell c_k will be moved from v_k to v_{n1}, and the vector properties v_k to v_{n1} will be updated.

- $\neg(c2v(c_k, v_{n1}) < c2v(c_k, v_k)) \wedge (c2v(c_{n1}, v_k) < c2v(c_{n1}, v_{n1}))$ $\rightarrow$ the cell c_n will be moved from v_{n1} to v_k, and the vector properties v_k to v_{n1} will be updated.
- $(c2v(c_k, v_{n1}) < c2v(c_k, v_k)) \wedge (c2v(c_{n1}, v_k) < c2v(c_{n1}, v_{n1}))$ $\rightarrow$ the cells c_k and c_{n1} will be switched, and the vector properties v_k to v_{n1} will be updated.

A random selection of the cells helps to avoid the risk of obtaining a final customization that depends on the specific cells comparison order.

At the end of this processing, each cell belongs to one, and only one, cluster in V. Collectively, the clusters will cover the entire interest area of processing. In Fig. B.2 cells belonging to the same cluster are drawn with the same color, depending on the cluster's properties vector, hence they appear like one single homogeneous area. The upper elaboration limit on the image is always corresponding to 50m, and is obtained frame-by-frame by a stereo based stabilization algorithm.

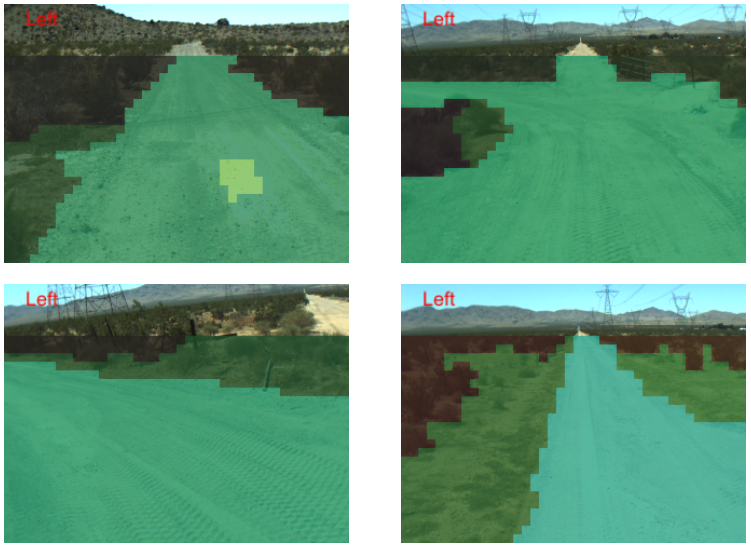


Figura B.2: Clustered images.

B.4 Decision

The decision phase tries to understand whether a cluster belongs to the road surface, using two different kinds of information:

- clusters properties as a *set of pixels*: homogeneity, size, shape factors.
- vehicle state: linear speed, linear acceleration, steering angle, steering angle acceleration.

The underlying idea is the following: each cluster belongs to the road with a given probability depending only on its own *intrinsic* properties: homogeneity (the higher the better), size (the higher the better), shape factors (the closer to road shape the better) and covered freespace area (the wider the better). The probability is computed at each frame without any memory of previous frame classifications. However having high probability of being road is not sufficient, and sometimes even *necessary*, to be finally classified as road. Suppose that the vehicle is running at the highest speed allowed by DARPA in a particular race segment, and is not steering and neither decelerating, but actually is running on the off-road surface. In this situation telling to the path planner that the road is *not* where the vehicle is driving could be extremely risky. This is why sometimes, even if a cluster has a low probability of being a road, the algorithm will classify it as a road and vice-versa. Total risk must be minimized and the risk associated with an incorrect classification depends on the current vehicle state. If the vehicle is running smoothly at high speeds, one should only deliver information about the location of the road which could alter vehicle behavior if one is very sure about the information. In other words, in the decision about a possible classification the following rule applies: classifications with higher risks require higher probabilities (of being correct) before they can be assigned.

A *Maximum Expected Utility* technique includes these risk considerations in the decision process. $U(S)$ denotes the utility of reaching state S as the consequence of a decision D. This utility function assigns a number to express the desirability of a state S. Instead of S, $R_i(D)$ can denote one of the possible outcome of the decision D.

Each outcome can occur with a probability $P(R_i(D)|D,E)$, where E summarizes the environment status in which the decision is taken. The *expected utility* of the decision D has the following value:

$$E(D) = \sum_{each R_i(D)} (U(R_i(D)) \cdot P(R_i(D)|D,E)) \tag{B.8}$$

In other words the expected value of taking a decision is the weighted sum of the possible utility, where the weights are the respective probabilities. A rational decider would take the action with the maximum expected utility.

In the context of the road classification decision, terms in (B.8) have the following interpretations:

- the possible decisions D are: classify a cluster as road or not road.
- the outcomes R_i are: the classification was correct or not correct.
- the environment E is determined by clusters properties and vehicle state.
- the utility function U depends on the consequences of the corresponding outcome on the vehicle.

Consequently the expected utility value can be expressed as follows:

$$E(road) = P(road) \cdot U_1 + (1 - P(road)) \cdot U_2 \tag{B.9}$$

$$E(off-road) = (1 - P(road)) \cdot U_3 + P(road) \cdot U_4 \tag{B.10}$$

where: U_1 is the utility of correctly classifying a cluster as road, U_2 is the utility of incorrectly classifying a cluster as road, U_3 is the utility of correctly classifying a cluster as off-road, U_4 is the utility of a incorrectly classifying a cluster as off-road.

The decision algorithm is implemented as a *cluster growing* process:

1. A cluster $c_k$from V is chosen randomly

2. If c_k is classified as off-road another cluster is chosen and the decision process returns to 1)

3. If c_k is classified as road with an expected utility E_k we define the current maximum expected utility $E_{max} = E_k$. Then:

   (a) We define c_{kgrown} the union of C_k and one of its neighbors not yet classified.

   (b) If c_{kgrown} is classified as off-road another neighbor of c_k is chosen and the process returns to a)

   (c) If c_{kgrown} is classified as road and if $E_{max} < E_{kgrown}$ then we merge C_k and its neighbor and process restarts from a).

4. When it is no longer possible to keep growing the road surface, adding new clusters, the process ends.

The mechanism of cluster growing implements a *one-shot* decision process, where any decision taken at time t does not influence the decisions taken at time $t+1$, and where decisions are taken only on the basis of the current environment and *current* possible outcomes. The following paragraphs discuss the computation of probability and utility functions.

B.4.1 Probability functions

To compute $P(R_i(D)|D,E))$ a complete causal model of the world is required. But having this causal model means having the solution to the path detection problem in general, because we were able to decide accurately when a cluster is actually road or not, by just image processing. Since we have to deal with a completely unknown environment we can only use an *estimation* of this probability distribution.

The probability function has the following form:

$$P(R_i(D)|D,E)) = \sum_{j=0}^{n} (p_j \cdot h_j) \tag{B.11}$$

where p_j is the probability that would be assigned if one only knew the $j-th$ cluster property (this $j-th$ cluster property is called y_j), with h_j the respective fixed weight. The underlying hypothesis is: the events $R_i(D)$, conditioned to the occurrence of D, y_j, are *disjoint events* when varying y_j. This allows the probability function to be expressed as a linear combination of simpler weighted probability functions.

Moreover the probability function might not involve the same sets of cluster properties when evaluating a cluster alone and when evaluating a set region already classified as road. In Section B.5 we will show an example.

B.4.2 Utility functions

The utility functions U_1, U_2, U_3, U_4 give a measure, in $[0,1] \in \Re$, of the desirability of actions' outcomes. Typically, correct classifications ($U1$,$U3$) have high utilities and incorrect classifications ($U2$,$U4$) have low utilities. However, in some circumstances safe operation might dictate the use of low values of $U3$ and high values of $U2$. These alternate weights increase the likelihood of classifying a given section as a road - even when the estimated probability that the section is, in fact, a road is low. High values of $U2$ (and $U1$) make the utility of classifying the section as a road large. Low values of $U3$ (and $U4$) make the utility of classifying the section as off-road small.

Each outcome has an associated set attributes, v_i, that determine its utility in terms of effect on the vehicle state. The set of attributes is assumed to be *mutually utility-independent (MUI)*[36]: a subset Σ of attributes is utility independent from another subset Γ if the preference between two outcome characterized by Σ_1, Σ_2 does not depend on the values of the corresponding Γ_1, Γ_2. In case of MUI the utility functions can be expressed in the following terms:

$$U = k_1 \cdot U_1 + k_2 \cdot U_2 + k_3 \cdot U_3 + ... + k_1 \cdot k_2 \cdot U_1 \cdot U_2 +$$

$$k_2 \cdot k_3 \cdot U_2 \cdot U_3 + k_1 \cdot k_3 \cdot U_1 \cdot U_3 + ... + k_1 \cdot k_2 \cdot k_3 \cdot U_1 \cdot U_2 \cdot U_3 + ...$$

where U_i is utility function of the the $i-th$ outcome attribute, and k_i the corresponding assigned weight.

B.5 Implementation

This paper has presented general frame-work, it will now present one specific implementation as an example.

B.5.1 Distance Function Implementation

The partial comparison functions showed in (B.1) (B.2) (B.3) are build on the HLS (Hue, Luminance, Saturation) color space. The images are originally acquired in Bayer pattern, and then converted into HLS just before the elaboration. The software acquisition layer also performs a selective gain control correction, maintaining the correct exposure for the obstacle/path detection area for each frame. No other preprocessing is applied on the images.

Cells properties vectors c_k are are built from the respective HLS pixel values (recall that each cell is a $d \cdot d = A$ group of pixels): $< (H_{k1},\ \ldots\ , H_{kA})\ ,\ (L_{k1}\ , \ldots,\ L_A)\ ,\ (S_{k1},\ \ldots\ , S_{kA}) >=< (x_{k1}^{[1]},\ \ldots\ , x_{k1}^{[A]})\ ,\ (x_{k2}^{[1]},\ \ldots\ , x_{k2}^{[A]})\ ,\ (x_{k3}^{[1]},\ \ldots\ , x_{k3}^{[A]}) >=< x_{k1}, x_{k2}, x_{k3} >$. Clusters properties vectors v_k are made of the corresponding cells' average HLS values: $< (\overline{H}_k, \sigma_{Hk}), (\overline{L}_k, \sigma_{Lk}), (\overline{S}_k, \sigma_{Sk}) >=< (\mu_{k1}, \sigma_{k1}), (\mu_{k2}, \sigma_{k2}), (\mu_{k3}, \sigma_{k3}) >=< y_{k1}, y_{k2}, y_{k3} >$.

Cell to cell (B.1) partial comparison functions take the following form:

$$D_{i-c2c}(x_{ki}, x_{li}) = \frac{\sum_{j=1}^{A} |x_{ki}^{[j]} - x_{li}^{[j]}|}{\sqrt{\sigma_{x_{ki}}^2 + \sigma_{x_{li}}^2}} \tag{B.12}$$

Cluster to cluster (B.2) partial comparison functions use cluster averages and variances:

$$D_{i-v2v}(y_{ki}, y_{li}) = \frac{|\mu_{ki} - \mu_{li}|}{\sqrt{\sigma_{ki}^2 + \sigma_{li}^2}} \tag{B.13}$$

Cluster to vector (B.3) partial comparison functions also use averages and variances:

$$D_{i-c2v}(c_{ki}, y_{li}) = \frac{|\overline{x_{ki}} - \mu_{li}|}{\sqrt{\sigma_{x_{ki}}^2 + \sigma_{li}^2}} \tag{B.14}$$

All the above functions are greater or equal than zero. Moreover, (B.12) and (B.13) are commutative, and to (B.13) also applies: $D_{i-c2c}(a,b) = 0 \iff (a = b)$.

B.5.2 Clusters properties

The clusters properties y_j used to compute the probability functions p_j in (B.11) are:

Name	Property
y_1	Average variance of cells' HLS values.
y_2	Number of cells (%).
y_3	Percentage of cells contained in the *freespace*.
y_4	Position and form factors 1.
y_5	Position and form factors 2.

Each property value is normalized to $[0,1] \in \Re$. The *freespace* is made of the portions of the image assumed to be free of obstacles on the basis of the stereo vision obstacle detector (Fig. B.3).

When searching for the first road cluster, the probability function depends only on the first four properties. The fifth property gets included after the initial road cluster gets classified. The refined position and form characterizations in the fifth property can help the cluster growing process to classify small complementary clusters.

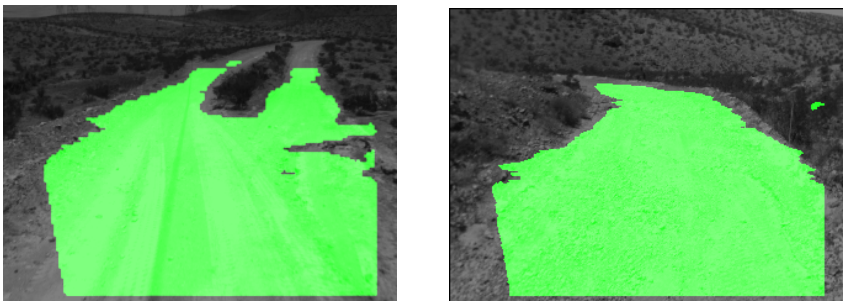


Figura B.3: Freespace obtained by the stereo vision: note that the system detect the whole drivable space in front of the vehicle.

B.5.3 Vehicle state

The following vehicle state parameters are used to compute the utility values and represent the outcome attributes:

Name	Property
v_1	Current truck speed.
v_2	Current steering angle.

The higher the speed the higher the risk related to change the current path. In Fig. B.4) is shown the bird's eye view map of the space in front of the vehicle, as classified by the path detector: green means road, red means off-road, black means unknown (out of the field of view). The white line represents the center of the detected road, while the blue line is the expected truck trajectory.

The farther the computed path is to the current trajectory, the higher the risk related to its classification. Hence the equation in B.4.2 becomes:

$$U = k_1 \cdot U_{v_1} + k_2 \cdot U_{v_2} + k_1 \cdot k_2 \cdot U_{v_1} \cdot U_{v_2}$$

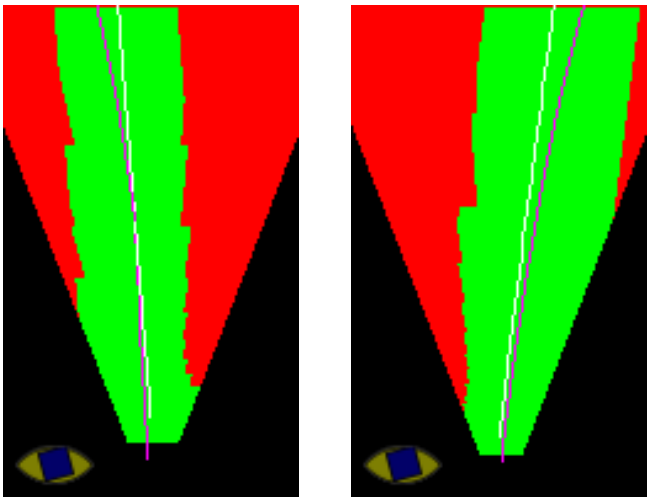


Figura B.4: The bird's eye view map of the space in front of the vehicle.

B.5.4 Decision Network Diagram

Fig. B.5 shows the decision network diagram of the frame-work. The ovals represent *chance nodes*: the random properties over which the decider has no influence (cluster properties). The rectangles represent *decision nodes*: properties influenced by the decider's choice (properties of the vehicle state). The diamonds represent the *utility functions*.

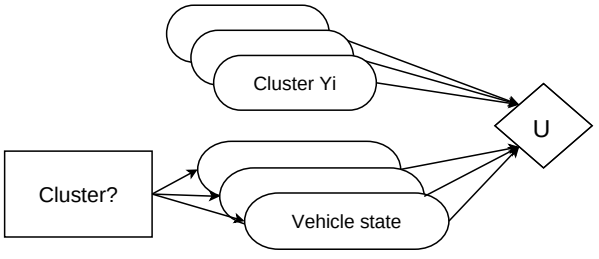


Figura B.5: Decision Network Diagram.

B.6 Conclusion and Future Works

The overall path detection algorithm elaboration of a single frame takes about 30ms using 320x240 pixels color images. Some classification results are shown in Fig. B.6.

This paper presented a frame-work based on clustering and decision theory to the path detection problem, specifically developed for the DARPA Grand Challenge 2005 and tested on the TerraMax^{TM} autonomous vehicle. The basic idea avoids the trap of starting with a very specific solution, which often works only in very specific conditions, and trying to somehow generalize it to cover all scenarios.

Instead, the approach presented here looks to divide the whole problem into a number of smaller tasks, which have been well studied in literature and for which effective algorithms have been developed. Collectively these tasks can solve the complete problem. This is useful especially in unstructured and very variable environments like the Grand Challenge 2005, where it is almost impossible to rely on any knowledge about the path. Additionally, accounting for the current vehicle state introduces some degree of high-level elaboration into the path detection problem, a typical medium-level task.

The next step will be improving the decision process allowing more complex reasoning. In particular a Markovian Decision Process could improve the decision process by minimizing the risk of converging to a local optimal solution. Moreover a great performance improvement would be given by having the GPS coordinates of waypoints that the vehicle must pass through in the future: in this way the path detector could deal with cross roads where more than one suitable path is visible, deciding which one has to be delivered to the vehicle manager on the basis of the expected trajectory, and also could deal with very sharp curves.

Another improvement could be obtained using a tracking step: a new clusters property, computed on the basis of the predicted most probable road surface at the next step, may be added to the list in B.5.2 and increase the decision process robustness.

The overall algorithm elaboration of a single frame takes about 30ms on a Pentium IV 3.0GHz using 320x240 pixels color images.

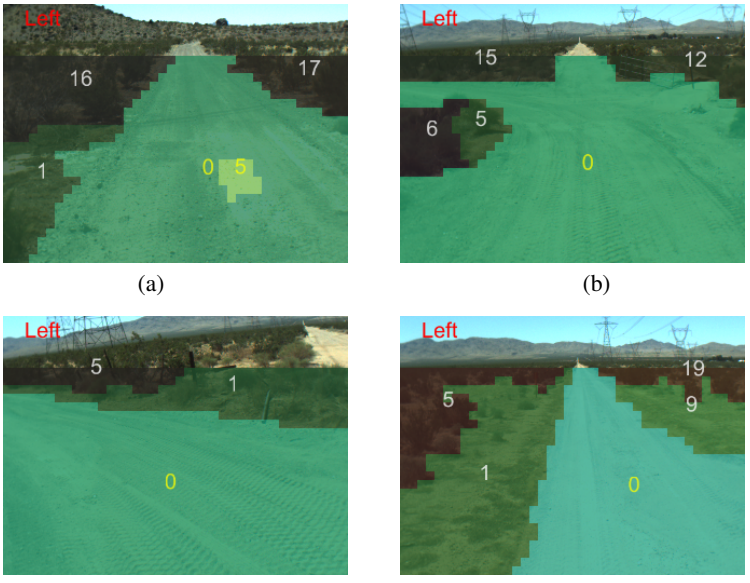


Figura B.6: Classification result: (a) the road is made of 2 clusters; (b) the truck is facing a junction; (c) a sharp curve; (d) a straight road. The color of the number super-imposed to clusters represents the classification outcome: white means off-road, yellow means road. White numbers denote clusters classified as off-road, yellow as road.

Appendice C

Lateral Vehicles Detection Using Monocular High Resolution Cameras on TerraMax^{TM}

Abstract

Autonomous driving in complex urban environments, including traffic merge, four-ways stop, overtaking, etc., requires a very wide range sensorial capabilities, both in angle and distance. This paper presents a vision system, designed to help merging into traffic on two-ways intersections, and able to provide a long detection distance (over 100m) for incoming vehicles. The system is made of two high resolution wide angle cameras, each one looking laterally (70 degrees) with respect of the moving direction, performing a specific background subtraction based technique, along with tracking and speed estimation. The system works when the vehicle is stopped at intersections, and is triggered by the high-level vehicle manager. The system has been developed and tested on the Oshkosh Team's vehicle TerraMax^{TM} , one of the 11 robots admitted to the DARPA Urban Challenge 2007 Final Event.

C.1 Introduction

After having seen five vehicles reaching the finish line in 2005, the Defense Advanced Research Project Agency (DARPA) moved its third-annual robot race Grand Challenge from the desert [37] into a city environment, calling it Urban Challenge [3]. The Urban Challenge features autonomous ground vehicles maneuvering in urban and sub-urban scenarios, where they had to execute merging into moving traffic, navigate traffic circles, negotiate busy intersections, avoid obstacles, follow lanes and handling parallel parking. Moving traffic was provided by several vehicles driven by professionals, as well as by the robots themselves, so robot-on-robot action was possible. Eighty-nine teams had applied to take part in the competition; DARPA accepted 35 of them, and only 11 were selected for the Final Event, competing for the $2 million first prize, $1 million second place prize, and $500,000 third place prize on a 55 miles course. The event took place on 3rd November 2007 at the former George Air Force Base, in California's high desert, on the area containing the abandoned officers' quarters.

This paper presents an artificial vision system developed for the Oshkosh Team's vehicle TerraMax™.The Oshkosh Team is a partnership between academic and industrial members: Oshkosh Truck Corporation, Teledyne Scientific Company (formerly known as Rockwell Collins Scientific), VisLab - University of Parma, Ibeo Automobile Sensor, and Auburn University. TerraMax was constructed from the chaiss of the same MTVR truck used by Marine Corps, removing the posterior axle to improve maneuverability, and fitting it with drive-by-wire technology and a computer network that hosts the software applications necessary for autonomous navigation and sensing.

The Artificial Vision and Intelligent Systems Lab of the University of Parma developed the artificial vision systems that sensed the environment. The vision system is composed of 11 cameras, able to provide the following sensing information: lane/path detection, stop line detection, obstacle detection for straight driving and maneuvering, backward vehicle detection for possible lane changing, oncoming traffic

detection when stopped at a stop line. The system that performed oncoming traffic detection is called “lateral system”, and it is discussed in this paper. Some details, thresholds, and numeric parameters are kept confidential, as they are proprietary information.

The paper is organized as follows: section C.3 presents the hardware architecture of the lateral vision system, section C.4 contains a detailed description of the detection algorithm, section C.5 discusses some experimental results and finally section C.6 draws the conclusions.

C.2 Lateral vision system constraints

When announcing the Urban Challenge, DARPA provided to the public the description of some typical situations the participants' vehicle were asked to deal with. One of the most challenging is the traffic merge, shown in Fig. C.1.

Assuming the maximum speed of vehicles in an urban environment is $13m/s$ (30mph), it is possible to deduce the *minimum* detection range needed for traffic merge like this: $13m/s * 10s = 130m$. Such a long distance is far beyond the typical LIDARs detection range. Moreover, the direction of oncoming vehicles falls outside of the field of view of the cameras used by the other vision systems, that are focused on the vehicle front and back. Hence, a specific vision system was needed. The constraints we had to meet are:

- long distance detection capabilities, intersections not always perpendicular;
- detection of *moving* oncoming objects, with speed estimation;
- real-time requirements must be met: 10Hz is the minimum processing rate.

The first constraints led us to choose high resolution and wide field of view cameras, in order to cover the several shapes an intersection can have, furthermore the cameras are mounted in a convenient position, as described in Sec. C.3. Since this

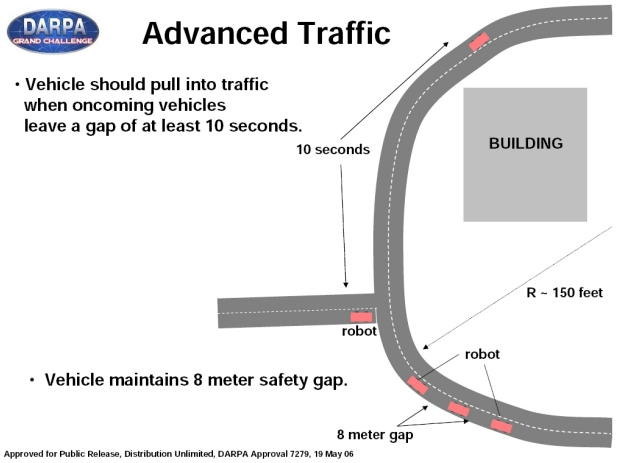


Figura C.1: Advanced traffic merge (Official DARPA document)

system is used at intersections when the vehicle is standing, the detection of moving objects is made via a *background subtraction* based detection, as described in Sec. C.4, along with tracking and speed estimation.

C.3 Hardware Setup

Fig. C.2 shows the hardware layout of the lateral system.

The cameras are equipped with a 1” sensor with a 1920x1080 pixels RGB resolution. As mentioned in Sec. C.2, the wider field of view, the better. Unfortunately, short focal length lenses for 1” sensors are too distorting for our purposes. After some tests we finally mounted 8mm Kowa lenses, that provide a good trade off between wide field of view and low distortion.

The cameras are mounted on a bar just in front of the vehicle, at a 165cm height from the ground, pointing 70 degrees away from the truck longitudinal axis.

If an object w meters wide at a distance of d meters is arriving at the intersection

where we are standing, we are able to estimate its size in pixes sp on the image as in the following:

$$sp = (focal\_length * w)/(d * pixel\_size) \tag{C.1}$$

so a generic vehicle ($w \simeq 1.5$m), 130m far from the truck, will be about 13 pixels wide on the image. Consequently the size of the smallest moving object the algorithm must be able to detect was set to 10 pixels.

Figura C.2: The two lateral cameras

C.4 Algorithm

C.4.1 Architecture

To deal with the large image size provided by the high resolution camera, without overrunning the strict time specifications described in Section C.1, we designed a *hybrid* multiresolution processing method. Two separate processings are applied on each captured image: the first one processes an horizontal slice of the full resolution image, while the second one operates on a downsampled version of the whole image. Each process is executed independently of the other one, and the results are collected and fused together. In this way the computing weight is considerably reduced, if

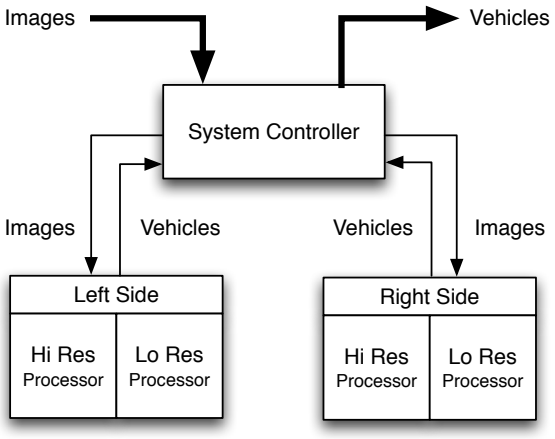


Figura C.3: System structure diagram

compared with a single full image processing, and at the same time we can still take advantage of the high resolution on far away zones, without precluding to check also near areas.

The multi resolution process is triggered by a specific network message, sent by the high-level vehicle manager: images are processed when the vehicle is in "stopped" state, where the speed is guaranteed to be 0. While, if the system do not receive any vehicle manager message nor inertial sensor data, it will keep computing in low resolution only; in this way the system is able to provide a minimum level of detection, at least at short distances.

Since the processings described above are totally independent of each other, in order to reduce the execution time we exploited the dual core CPU, by executing the two processings in separate threads. This structure is thus replicated on each side, right and left, and both are put in execution again on separate threads; the final high level system structure diagram is represented in Fig. C.3.

It is important to notice that all the process is done on gray images, ignoring the color information to keep the computing time low.

C.4.2 Background Subtraction

The basic layer of the system is a *background subtraction* algorithm [11] [12], implemented with some additional features, in order to overcome some problems encountered during the development. Even if the algorithm is executed only when the truck is stopped, the engine vibrations are so strong to make the camera oscillate (estimated in more than ± 5 pixels vertically), causing non-zero differences when subtracting background objects. Moreover, in a complex urban scenario many objects can change their appearance periodically, like flags, trees and also traffic lights. Both vibrations and periodic movements can cause false positives detection. While the second problem can be solved in the tracking step (see Section C.4.5) by filtering out objects that do not change their average position in time, the first one is more subtle. Due to vertical oscillations, the non-zero responses in the difference image often appear like wide horizontal edges, hard to uniquely be located in world coordinates: this prevents the tracker from filtering them out on the basis of a persistent position in time.

For these reasons, the vibration problem needed an ad-hoc solution: the first step of the algorithm is a per-pixel difference between the last acquired frame and a reference image, built as the time weighted *average* of the previous N frames.

The weights must be computed in order to assign higher importance to recent frames against far ones; furthermore they have to preserve the global image brightness. We can then build the desired weights array, with a linear decreasing curve, by computing a k factor satisfying the following equation:

$$k \cdot \sum_{i=0}^{N-1} (N-i) = 1 \tag{C.2}$$

where N is the number of frames considered during the operation; so, according to equation (C.2) the resulting weights are all the multiples of the constant k computed in this way:

$$k = \frac{2}{N(N+1)}. \tag{C.3}$$

Using these weights, the j-th pixel of the reference image is computed as:

$$I_{ref}[j] = \sum_{i=0}^{N-1} k \cdot i \cdot F_{t-i}[j] \tag{C.4}$$

where F_{t-i} are the previous acquired frames.

By applying these operations, the image we obtain is a pseudo-background that can be used to reveal movements, with the important features of noise-reduction and high vibrations attenuation; we have though to notice that N (the number considered frames) should not be too high, to avoid too strong blurring effects, that may lead to trace moving objects impressed on the reference image for too long, and to a deterioration of the detection performance.

Now it is possible to compute the absolute difference pixel-by-pixel, and immediately after, to apply a threshold to quantize the resulting image in two levels:

$$I_{diff} = \begin{cases} 255 & \text{if } \left|I_t - I_{ref}\right| \leq threshold \\ 0 & \text{otherwise} \end{cases} \tag{C.5}$$

In Fig. C.4(c) the final result of this stage is shown: white ones corresponding to moved points and the black ones to fixed points on the image.

C.4.3 Detection

Still focusing on the aim of keeping the vehicle detection process simple and computationally light, due to time constraints, all zones revealed by the difference are not bounded by a labeling algorithm; they are instead enclosed into bounding boxes with a simpler histogram approach, based on the assumption that there could not be stacked moving vehicles. Hence, to reveal all the zones in which there has been some movement, we make a vertical histogram of the binarized difference image, counting for each column the number of white pixels; then we create several slices of the image, one for each zone where the histogram is higher than a threshold (Fig. C.4(d)).

Once we have obtained the vertical slices, the upper and the lower bounds of the

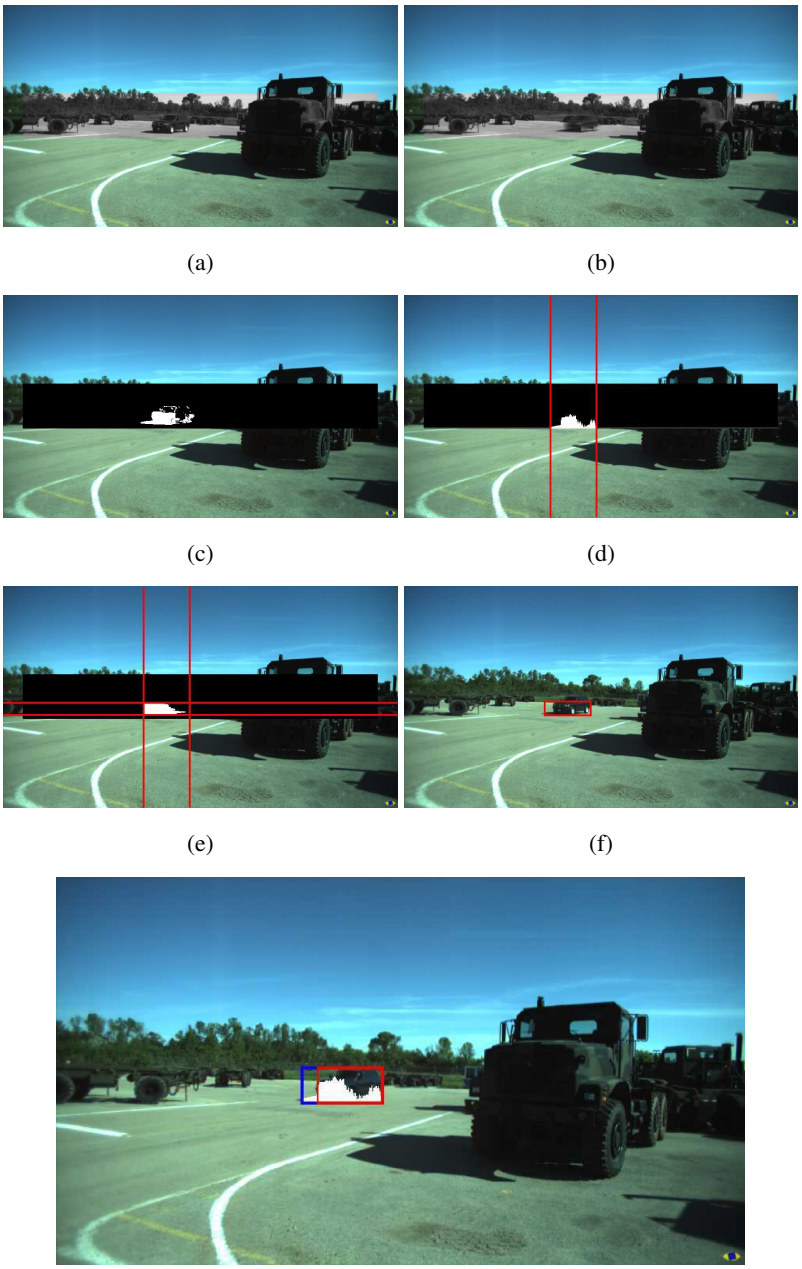
moving object are searched inside those, by computing a new histogram: this time it will be horizontal, looking for the first and the last row where the histogram is over the threshold. Finally the horizontal coordinates of the slices, and their upper and lower bounds, are used to create a bounding box for each moving object (Fig. C.4(e)).

A further operation we apply to the created bounding boxes is a shadow eraser filter: this because also the pixels corresponding to the shadows projected by vehicles on the ground are, at this stage, seen as moving objects. In low sun conditions these shadows may cause an error in position estimation of more than 1 meter. Since the obstacles provided by the lateral system have to be fused together with other sensors' output, it is important to be as much accurate as possible, to do not introduce noise into the higher-level fusion step. To remove shadows we suppose that the height of a lateral projected shadow of a vehicle is smaller than the vehicle itself; hence the shadow eraser filter analyzes the slice of the histogram relative to each bounding box, and determine if there are zones, at both sides of the slice, which have an histogram height heavily lower than the box's height; in that case the filter will cut that part, as shown in Fig. C.4(f)(g) which presents how moving objects are shown at this preliminary stage.

C.4.4 Progressive Background Generation

We have seen how motion detection is based on a background subtraction algorithm, but we have also shown how that step depends on the difference between two images; the difference operation is able to enhance pixels where the image has changed, but only on the edges of the moving object.

The proposed solution tries to replace the reference image, as computed in equation (C.4), with the background of the scene as it would appear if no moving objects were present. This can be easily achieved when working in a-priori known environments, like in traffic monitoring or in laboratory testing, just taking manually the best background image, and using it indefinitely. Unfortunately in our case this is not possible, since the cameras are installed on a moving vehicle, hence we have studied a method to *progressively* generate the background image every time the vehicle stops



(a) (b)

(c) (d)

(e) (f)

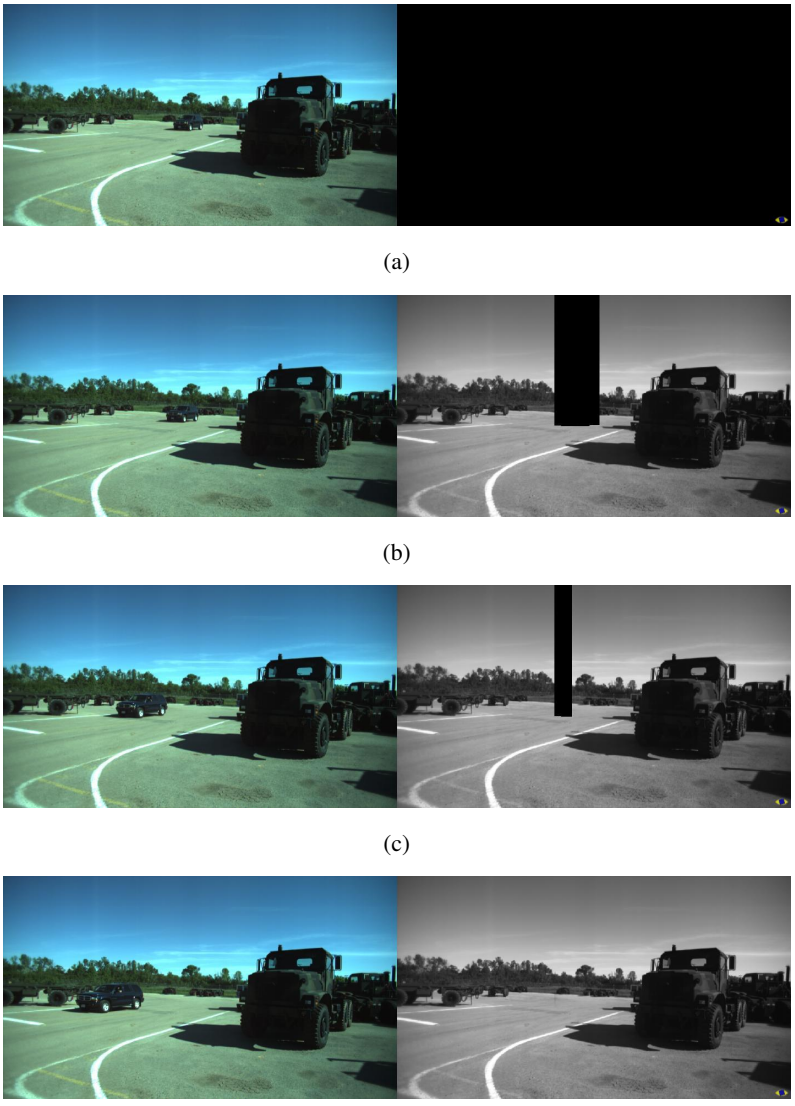
(g)

Figura C.4: Preliminary background subtraction and detection: (a) the last captured image, (b) the reference image, (c) the binarized difference, (d) the vertical histogram, (e) the horizontal histogram made on the slice computed on the basis of the vertical histogram, (f) the bounding box including the vehicle's shadow and (g) the final detection result.

at an intersection, working just on the acquired stream. The basis of the background generation stands on the assumption that vehicles in the scene are in motion, so they will not hide the same part of the scene's background forever.

The starting point of the algorithm is the list of bounding boxes created by the detection phase and an empty background image (Fig. C.5 (a)). At each execution of the process, one per acquired frame, the black zones of the background image are updated in this way: if their corresponding pixels are now not hidden by some moving object (and this information is easy to retrieve by using the list of bounding boxes), they are filled with the pixels of the input image.

It is clear how this process is iterative and incremental (Fig. C.5 (b),(c),(d)) and takes a couple of frames to complete, depending on the speed of vehicles. However, it is possible to take advantage of having also a partial background, by using it immediately with just some adjustments: it is possible to substitute the reference image with the background image during the background generation process. This approach is very efficient, but still keeps all noise and vibrations which affect the input image; hence we decided to use again the reference image, instead of the input one, for the background image filling. In this way we can take advantage of the generated background image and, at the same time, of the vibration attenuation effect given by the reference image, obtaining a better defined obstacles' shape. Finally, once the background is completed, it can be considered frozen and no more filling are needed. The only problem is when a vehicle that was initially static starts moving; this situation can create a standing "ghost" on the system output. In fact, once the started vehicle has completely detached from its initial position, the binarized image will presents two blobs: one corresponding to the real moving vehicle and one corresponding to the shape of the same vehicle, in the background image. In this case the higher level tracking algorithm will discard the standing ghost, since it is not moving at all, and its image area will be replaced by the corresponding area in the reference image.



(a)

(b)

(c)

(d)

Figura C.5: The progressive background generation process. In (a) the background is empty. In (b)(c) how the background is generated as moving objects change their position. In (d) the final background.

C.4.5 Tracking

In order to stabilize the algorithm output, a tracking stage is required to reduce false positives/negatives and, at the same time, to estimate vehicles' speed by exploiting their history. The simplicity guideline has affected also this part of the algorithm, leading up to develop a simple tracking system which working on the positions of the detected bounding boxes, instead of using other heavy feature-based systems, well known in literature [38]. The developed tracking system operates in two steps: the motion analysis phase, to search for a mapping between the list of previously detected (and tracked) vehicles and the list of the new bounding boxes, and a filtering phase, whose goal is to discard all objects with a motion reckoned incompatible to a vehicle's one.

C.4.5.1 Motion Analysis

As introduced above, with motion analysis we want to find the relations between the existent list of tracked obstacles and the current bounding boxes, finding for each previously known vehicle its new position. The approach we adopted to create these relations is based on the assumption that the bounding boxes belonging to a single object, in two adjacent frames, must have an overlapped area. Furthermore we assume that the most relevant point of the bounding box belonging to a vehicle is the lower left corner for the right camera images, or the lower right for the left camera (see Fig. C.4(g)). We chose those points because they probably correspond to the closest part of an approaching vehicle and, at the same time, they are the most accurate points we can consider belonging to the vehicle, due to the monocular vision limits. Hereinafter with linchpin corner we will refer to that points.

Under these assumptions we can assert that, if a bounding box of the previous list has an overlapped area with one, and only one, of the current boxes, they can be associated; if the considered box is overlapped with several current boxes, we can choose the one with the nearest linchpin corner as the most appropriate. When the associations are created, each obstacle tracking information is updated with its new

position and its last motion vector, calculated between the current linchpin corner and its previous one. The other cases, like vehicles arrival/departure or intermittent detections, are handled in part with an hysteresis window put on the acceptance/discard phase, like any other tracking algorithm, and in part by the filtering phase of the tracker (see Section C.4.5.2).

C.4.5.2 Filtering

The filtering phase of the tracker works on the statistics data of each obstacle, calculated on a maximum time window of 20 frames, in particular on its motion vector's average length and its direction's variance; these two parameters are very important in order to determine if the motion of a tracked object is compatible with a vehicle's one. In fact, by analyzing the average length of the motion vector, it is possible to figure out if a detected object is moving or not: if the average length is equal to 0, we are in the case of still object and we can assume that the obstacle may be a false positive, like the "ghost" obstacles discussed in paragraph C.4.4, and it can be discarded as well.

The variance of the motion vector's direction is, instead, used to infer some information about object's motion regularity; in fact a small value for that index will correspond to a uniform movement, the one we expect from a vehicle which is moving upon a street, while a greater variance will correspond to a quite random motion, maybe belonging to some swinging vegetation or similar. So all these considerations about motion statistics are represented by a score assigned to each object, that is increased/decrease according to the statistic indexes analysis; finally the score is used to update the status of an obstacle, that can be one of the following: *approved*, *keeping*, and *discarded*.

C.4.6 Coordinates conversion

The part of the algorithm explained so far shows the whole process we used to identify vehicles on images, but a further step is needed to provide the real world position

of the detected vehicles to the higher level step. Hence the pixel coordinates of each obstacle must be converted into TerraMax^{TM}reference frame, set on the center of the front bumper projected on the ground. The approach we used to this conversion, is an Inverse Perspective Mapping (IPM) [39]: assuming to know all the camera's parameters, the IPM allows to convert image coordinats to world coordinates by applying a geometrical transformation.

Once we have assigned the position expressed in meters to each detected moving vehicle, the final step will be to send them to the appropriate higher level service, the world perception server, so they can be used by the navigation system.

Moreover, the world position of the vehicles is also used to estimate their speed, by considering the ratio between the traveled distance and the time elapsed.

However the coordinate conversion stage is very critical, because of the high noise sensitivity of the IPM, especially at great distances (like 100m). In fact an error of a single pixel, during the calibration process, can be reflected on an error of several meters in the obstacles' position determining.

C.5 Results

The algorithm showed good detection rate during test sessions, especially with respect to far vehicles; however some false positives/negatives are present, even if the introduction of the tracking system has drastically reduced their number. As far as time is concerned, the overall system takes less than 100ms on an Intel Core Duo 2.0GHz (T2500) to process both right and left images in multi resolution mode. Some results are shown in Fig. C.6 and Fig. C.7.

C.6 Conclusions and Future Works

This paper presented a moving vehicle/object detector, using monocular high-resolution cameras and based on an enhanced background subtraction technique. The system was developed and tested on the Oshkosh Team's vehicle TerraMax^{TM} and was spe-

cifically designed to help it during traffic merge manoeuvres at the DARPA Urban Challenge 2007. The main difficulty of this task was the detection of long distance oncoming traffic, due to the pixel size a vehicle has at such distances and to the high vibrations affecting the cameras.

The lateral system demonstrated great potential during the TerraMax™ development and testing: it was able to accurately detect moving objects further that 100m, to estimate their speed and met the real-time constraints. However some improvements can be applied to this system, in order to overcome basically two issues: the high sensitivity to camera calibration parameters during the vehicles' position computation, and the fact that currently the system does not discriminate between vehicles, pedestrians and others. The first one can be mitigated by exploiting inertial information provided by the INS system, and apply an on-line correction to calibration parameters; while the latter could be solved by introducing an object classification system to notify which objects have to be considered and which have to be discarded.

(a) (b)

(c) (d)

(e) (f)

Figura C.6: Detection results. For readers convenience, the left side shows the complete result images and the right side the same images zoomed in to highlight far away detected objects.

(g) (h)

(i) (l)

(m) (n)

Figura C.7: More detection results. For readers convenience, the left side shows the complete result images and the right side the same images zoomed in to highlight far away detected objects.

Bibliografia

[1] Defence Advanced Research Projects Agency (DARPA). Grand Challenge 2004. Avaiable http://www.darpa.mil/grandchallenge04.

[2] Defence Advanced Research Projects Agency (DARPA). Grand Challenge 2005. Avaiable http://www.darpa.mil/grandchallenge05/.

[3] Defence Advanced Research Projects Agency (DARPA). Urban Challenge 2007. Avaiable http://www.darpa.mil/grandchallenge.

[4] Alberto Broggi, Stefano Cattani, Pier Paolo Porta, and Paolo Zani. A Laserscanner-Vision Fusion System Implemented on the TerraMax Autonomous Vehicle. In *International Conference on Intelligent Robots and Systems (IROS06)*, Beijin, China, October 2006.

[5] Alberto Broggi, Claudio Caraffi, Pier Paolo Porta, and Paolo Zani. The Single Frame Stereo Vision System for Reliable Obstacle Detection used during the 2005 Darpa Grand Challenge on TerraMax. In *IEEE Intelligent Transportation System Conference (ITSC06)*, Toronto, Canada, September 2006.

[6] A. Dahanayake, H. Sol, and Z. Stojanovic. Modeling and design of service-oriented architecture. *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, 5.

[7] Michael P. Papazoglou and Willem-Jan Van Den Heuvel. Service-oriented design and development methodology. *International Journal of Web Engineering and Technology*, 2006.

[8] Mauro Birattari, Gianni Di Caro, and Marco Dorigo. Toward the formal foundation of ant programming. In *Ant Algorithms*, pages 188–201, 2002.

[9] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.

[10] I. Matzkevich and B. Abramson. Decision analytic networks in artificial intelligence. *Management Science*, 41:1–22, 1995.

[11] R. Cucchiara, C. Grana, M. Piccardi, and A. Prati. Detecting moving objects, ghosts, and shadows in video streams. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(10):1337–1342, 2003.

[12] Anurag Mittal and Nikos Paragios. Motion-based background subtraction using adaptive kernel density estimation. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'04)*, 02:302–309, 2004.

[13] Umit Ozguner, Keith A. Redmill, and Alberto Broggi. Team TerraMax and the DARPA Grand Challenge: A General Overview. In *Procs. IEEE Intelligent Vehicles Symposium 2004*, pages 232–237, Parma, Italy, June 2004.

[14] Massimo Bertozzi and Alberto Broggi. GOLD: a Parallel Real-Time Stereo Vision System for Generic Obstacle and Lane Detection. *IEEE Transactions on Image Processing*, 7(1):62–81, January 1998.

[15] Massimo Bertozzi, Alberto Broggi, and Alessandra Fascioli. Vision-based Intelligent Vehicles: state of the art and perspectives. *Journal of Robotics and Autonomous Systems*, 32(1):1–16, June 2000.

[16] Christopher Urmson, Joshua Anhalt, Michael Clark, Tugrul Galatali, Juan Pablo Gonzalez, Jay Gowdy, Alexander Gutierrez, Sam Harbaugh, Matthew Johnson-Roberson, Hiroki Kato, Phillip L Koon, Kevin Peterson, Bryon K Smith, Spencer Spiker, Erick Tryzelaar, and William Red L. Whittaker. High speed navigation of unrehearsed terrain: Red team technology for grand challenge 2004. Technical Report CMU-RI-TR-04-37, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, June 2004.

[17] Young Uk Yim AND Se-Young Oh. Three-feature based automatic lane detection algorithm (TFALDA) for autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 4:219–225, 2004.

[18] S. Goss, S. Aron, J. L. Deneubourg, and J. M. Pasteels. Self-organized shortcuts in the argentine ant. *Naturwissenshaften*, 76:589–581, 1989.

[19] Osvaldo Gómez and Benjamín Barán. Reasons of ACO's Success in TSP. In *ANTS Workshop*, pages 226–237, 2004.

[20] Kwang Mong Sim and Weng Hong Sun. Ant colony optimization for routing and load-balancing: survey and new directions. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 33(5):560–572, 2003.

[21] Anne Wade and Said Salhi. An ant system algorithm for the mixed vehicle routing problem with backhauls. *Metaheuristics: computer decision-making*, pages 699–719, 2004.

[22] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Intreduction*. MIT Press, Cambridge, MA, 1998.

[23] Walter J. Gutjahr. Aco algorithms with guaranteed convergence to the optimal solution. *Inf. Process. Lett.*, 82(3):145–153, 2002.

[24] Thomas Stützle and Marco Dorigo. A short convergence proof for a class of ant colony optimization algorithms. *IEEE Trans. Evolutionary Computation*, 6(4):358–365, 2002.

[25] Alberto Broggi, Massimo Cellario, Paolo Lombardi, and MArco Porta. An evolutionary approach to visual sensing for vehicle navigation. *IEEE Transactions on Industrial Electronics*, 50, February 2003.

[26] Rockwell Collins Oshkosh Truck Corporation and University of Parma. Team TerraMax 2005. Avaiable http://www.terramax.com.

[27] R. Aufrère, R. Chapuis, and F. Chausse. A fast and robust vision based road following algorithm. In *Procs. IEEE Intelligent Vehicles Symposium 2000*, pages 192–197, Detroit, USA, October 2000.

[28] Christopher Rasmussen. Combining Laser Range, Color, and Texture Cues for Autonomous Road Following. In *IEEE International Conference on Robotics and Automation*, pages 4320–4325, Washington, DC, USA, May 2002.

[29] Alberto Broggi and Stefano Cattani. An Agent Based Evolutionary Approach to Path Detection for Off-road Vehicle Guidance. *Special Issue on Evolutionary Computer Vision and Image Understanding, Pattern Recognition Letters*, 27:1164–1173, August 2006.

[30] D. Mateus, G. Avina, and M. Devy. Robot Visual Navigation in Semi-structured Outdoor Environments. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 4691–4696, Barcelona, Spain, April 2005.

[31] N. B. Karayiannis and M. M. Randolph-Gips. Soft learning vector quantization and clustering algorithms based on non-Euclidean norms: Single-norm algorithms. In *IEEE Transactions on Neural Networks*, volume 16, pages 423–435, 2005.

[32] P. H. Batavia and S. Singh. Obstacle Detection Using Adaptive Color Segmentation and Color Stereo Homography. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 705–710, Seoul, Korea, May 2001.

[33] Tae-Kyun Kim and Josef Kittler. Locally linear discriminant analysis for multimodally distributed classes for face recognition with a single model image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(3):318–327, 2005.

[34] J.Q. Smith. *Decisions Analisys*. Chapman and Hall, London, 1988.

[35] Alberto Broggi, Claudio Caraffi, Pier Paolo Porta, and Paolo Zani. The Single Frame Stereo Vision System for Reliable Obstacle Detection used during the 2005 Darpa Grand Challenge on TerraMax. In *IEEE Intelligent Transportation System Conference (ITSC06)'*, Toronto, Canada, September 2006.

[36] R.L. Keeney, J.S Hammond, and H. Raiffa. *Smart choices: A guide to making better decisions*. Harvard University Press, Boston, 1999.

[37] Claudio Caraffi and Stefano Cattani. VisLab at the Grand Challenge. *IEEE Computer*, 39(12):36–37, December 2006.

[38] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, Carnegie Mellon University, April 1991.

[39] Massimo Bertozzi, Alberto Broggi, and Alessandra Fascioli. Stereo Inverse Perspective Mapping: Theory and Applications. *Image and Vision Computing Journal*, 8(16):585–590, 1998.