UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
ENGENHARIA ELETRÔNICA

Vitor de Castro Carvalho

Análise e Implementação de um Cinturão de Radares para Carros Autônomos

Florianópolis
2020

Vitor de Castro Carvalho

Análise e Implementação de um Cinturão de Radares para Carros Autônomos

Trabalho de Conclusão de Curso de Graduação em Engenharia Eletrônica do Centro Tecnológico da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Engenharia Eletrônica.
Orientador: Prof. Carlos Aurélio Faria da Rocha, Dr.

Florianópolis
2020

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Carvalho, Vitor de Castro
Análise e Implementação de um Cinturão de Radares paraCarros Autônomos / Vitor de Castro Carvalho ; orientador, Carlos Aurélio Faria da Rocha, 2020.
39 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Eletrônica, Florianópolis, 2020.

Inclui referências.

1. Engenharia Eletrônica. 2. Sistemas Embarcados. 3. Processamento de Sinais. 4. Veículos Autônomos. 5. Radar. I. da Rocha, Carlos Aurélio Faria . II. Universidade Federal de Santa Catarina. Graduação em Engenharia Eletrônica. III. Título.

Vitor de Castro Carvalho

Análise e Implementação de um Cinturão de Radares para Carros Autônomos

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de Bacharel em Engenharia Eletrônica e aprovado em sua forma final pelo Curso de Engenharia Eletrônica.

Florianópolis, 18 de dezembro de 2020.

Documento assinado digitalmente

Fernando Rangel de Sousa
Data: 18/12/2020 13:13:17-0300
CPF: 884.649.114-91

Prof. Fernando Rangel de Sousa, Ph.D.
Coordenador do Curso

Banca Examinadora:

Documento assinado digitalmente

Carlos Aurelio Faria da Rocha
Data: 18/12/2020 13:04:18-0300
CPF: 060.075.032-91

Prof. Carlos Aurélio Faria da Rocha, Dr.
Orientador

Documento assinado digitalmente

Eduardo Augusto Bezerra
Data: 21/12/2020 09:24:57-0300
CPF: 830.851.577-00

Prof. Eduardo Augusto Bezerra, Ph.D.
Avaliador
Universidade Federal de Santa Catarina

Documento assinado digitalmente

Leonardo Silva Resende
Data: 18/12/2020 15:01:09-0300
CPF: 524.735.206-82

Prof. Leonardo Silva Resende, Ph.D.
Avaliador
Universidade Federal de Santa Catarina

Este trabalho é dedicado aos meus queridos pais.

AGRADECIMENTOS

Expresso meus agradecimentos aos meus pais, que sempre me deram incondicional apoio e me permitiram uma dedicação exclusiva aos estudos.

Agradeço ao meu tutor na França, Cedric Dulau, pela experiência enriquecedora do estágio.

Estendo meus agradecimentos a todo o corpo docente do Departamento de Engenharia Elétrica e Eletrônica pelo ensino de qualidade provido, especialmente ao professor Carlos Aurélio pela ajuda especial nessa caminhada, tanto antes, quanto após meu intercâmbio.

Por último, mas não menos importante, gostaria de agradecer aos servidores da Universidade de Santa Catarina por tornarem possível o funcionamento deste ensino de qualidade e sobretudo público.

RESUMO

Durante a última década, vários estudos estatísticos apontaram que as principais causas dos acidentes de trânsito tem como ponto em comum a ação humana. A fonte dos possíveis problemas normalmente reside em falhas na percepção do condutor com relação ao ambiente em volta do carro e, normalmente, essa perda de informação induz o motorista a tomar decisões ruins. Este trabalho visa, então, a implementação de um sistema que permitirá ao veículo ter uma percepção de 360°do ambiente em sua volta por meio de um cinturão de radares. Após a realização da arquitetura do sistema embarcado, realizou-se a integração deste sistema em um veículo real. No final, foi possível ao veículo, por meio de processamento do sinal radar, detectar objetos estáticos e dinâmicos e consequentemente calcular as zonas de espaço livre, por onde ele poderia passar.

Palavras-chave: Radar. Sistemas Embarcados. Processamento de Sinais. Veículos Autônomos

ABSTRACT

Over the last decade, several statistical studies have found that the main causes of the car accidents over the world are related to the driver. The problems might be due a lack of perception of the driver aiming him do take bad decisions. Therefore, this work aims to build a system which will allow the vehicle to have a 360°environment perception through a radar belt. After the embedded system architecture was done, this system was integrated in a real vehicle. At the end, the vehicle was able to detect static and dynamics objects and to calculate the free space around it through some radar signal processing.

Keywords: Radar. Embedded Systems. Signal Processing. Autonomous Vehicles.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ADAS	Advanced Driver Assistance Systems
ARS430	Advanced Radar Sensor 430
CAN	Controller Area Network
DBC	DataBase CAN
eCAL	enhanced Communication Abstraction Layer
PTP	Precision Time Protocol
RCS	Radar Cross Section
VDY	Vehicle Dynamics

SUMÁRIO

1 INTRODUÇÃO

Atualmente, o homem é a principal causa dos acidentes de trânsito. Seja nas estradas ou nas ruas de cidades, muitas vezes falta ao condutor detalhes sobre a percepção do ambiente em sua volta e as mudanças que ocorrem nele a todo momento, levando-o a ter uma interpretação equivocada e tomar decisões que acabam por causar os acidentes.

Nesse contexto, foi criado uma área de desenvolvimento da indústria, conhecida pela sigla Advanced Driver Assistance Systems (ADAS), para tornar os carros mais inteligentes. Integrando vários tipos de sensores ao redor do carro, os engenheiros pretendem criar sistemas capazes de informar ao condutor detalhes difíceis de serem percebidos, dar assistência em situações de perigo ou de extrema necessidade e, por fim, conduzir-se sozinhos.

Um dos sensores mais utilizados nessa indústria é o radar. Com o envio e recepção de pulsos eletromagnéticos, é possível determinar obstáculos com precisão em termos de distância e direção. Por sua natureza eletromagnética, ele consegue detectar objetos muito distantes e funciona muito bem em condições adversas de estrada, como chuva, neve ou névoa.

1.1 OBJETIVOS

1.1.1 Objetivos Gerais

Este trabalho tem como objetivo de propor uma solução para integrar um sistema de seis radares em torno de um carro capaz de detectar objetos dinâmicos e calcular zonas de espaço livre; e, por fim, implementar este sistema em um veículo real.

1.1.2 Objetivos Específicos

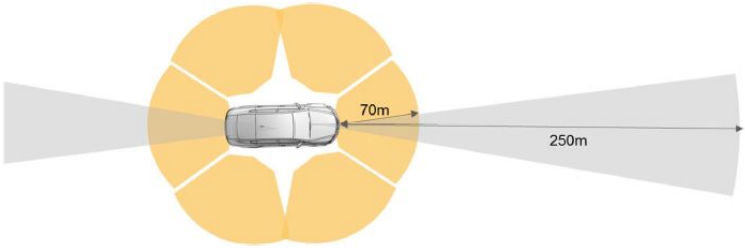
1. Determinar a arquitetura hardware para integrar o sistema no veículo;
2. Configurar o switch responsável pela comunicação entre os radares e a unidade central de processamento de dados;
3. Configurar a unidade central de processamento de dados que será embarcada no veículo;
4. Estudar os protocolos internos de comunicação desenvolvidos pela Continental, que permitem a comunicação entre todos os módulos do sistema;
5. Configurar um roteador PCAN, responsável por recuperar os dados internos do veículo através do bus Controller Area Network (CAN);
6. Embarcar todos os dispositivos no veículo, conectá-los e se certificar de que o sistema funciona;
7. Coletar os dados do sistema funcionando em diversas situações reais de trânsito, certificar-se que a latência é baixa o suficiente para aplicações de tempo-real, analisar e validar os resultados.

2 O CINTURÃO DE RADARES

A integração do Cinturão de Radares pode ser considerada como um sub-projeto dentro de um projeto maior chamado *DemoCar*, que tem como objetivo utilizar diferentes tipos de sensores para captar informação do ambiente ao redor do veículo.

2.1 APRESENTAÇÃO

Figura 1 – Percepção de 360°com 6 radares ARS430



O Cinturão de Radares é um sistema protótipo de múltiplos radares posicionados de maneira ideal para obter-se uma visão sem pontos-cegos (360°) em volta do veículo.

Cada um dos radares faz dois *scans* por ciclo, pois cada *scan* possui características distintas. Um consegue detectar objetos até mais longe (250m), porém possui um ângulo de visão mais estreito (18°). O outro consegue detectar objetos apenas até 70m, porém tem um campo de visão maior, variando entre 90° e 150°. O período de cada um desses ciclos corresponde a 60ms e o radar emite ondas eletromagnéticas a uma frequência de 77GHz.

Esses radares contêm um pré-processamento interno sobre a

reflexão recebida das ondas eletromagnéticas emitidas, tendo, então, em sua saída uma lista de detecções. Cada objeto desta lista representa uma detecção e contém uma série de características sobre tal detecção:

a) Distância radial (R) [m];

b) Velocidade radial relativa (V_R) [m/s];

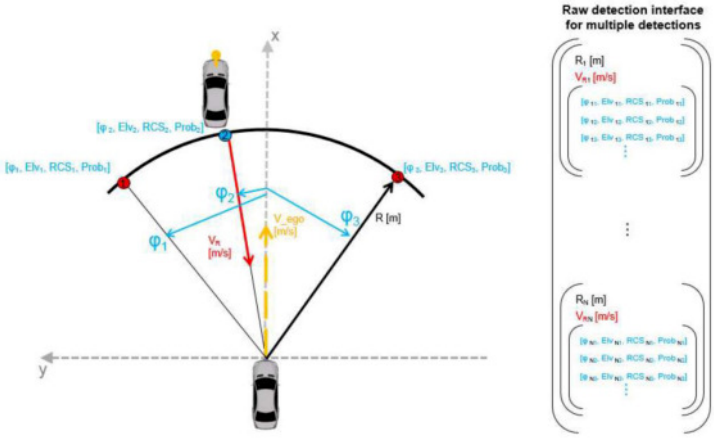
c) Probabilidade de ambiguidade ($Prob$) [%];

d) Ângulo azimutal (φ) [°]

e) Radar Cross Section (RCS)

A representação gráfica dessas características, assim como o modelo da lista de saída do radar, pode ser vista na Figura 2.

Figura 2 – Representação gráfica da lista de saída do radar ARS430



Com o Cinturão de Radares funcionando e calibrado fica possível rodarmos os algoritmos de alto nível, que são o *Rastreador de Objetos Dinâmicos*, que permite ao veículo detectar objetos "vivos", como

pedestres, ciclistas e outros carros; e o *Mapa de Ocupação/Detector de Espaço Livre* que faz o mapeamento dos objetos estáticos ao redor do carro e subsequentemente calcula o espaço livre onde o carro pode passar, informação primordial para que um carro seja autônomo.

3 INTEGRAÇÃO DO CINTURÃO DE RADARES

3.1 MOTIVAÇÃO

Para a fabricação de carros autônomos, a fusão de dados de diferentes tipos de sensores é imprescindível, pois cada tipo tem suas particularidades, apresentando vantagens e desvantagens em diferentes casos.

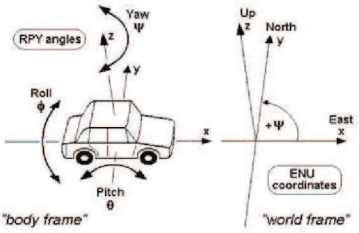
Os radares em particular possuem muitas vantagens e atualmente estão presente mesmo em carros comuns, para dar uma assistência à condução do motorista. Dentre suas vantagens, podemos listar o baixo custo de fabricação, tecnologia bem conhecida e consolidada, tamanho de dados pequeno e, finalmente, boa performance em condições adversas de estrada, como neve, chuva ou neblina.

Sendo assim, esse conjunto de seis radares em volta do carro permite ao sistema obter uma ferramenta poderosa para detecção de objetos em longas distâncias, mesmo em condições adversas.

3.2 INSTALAÇÃO DOS RADARES NO CARRO DE TESTE

Na área automobílistica há uma convenção para o sistema de coordenadas. Este sistema está ilustrado na Figura 3.

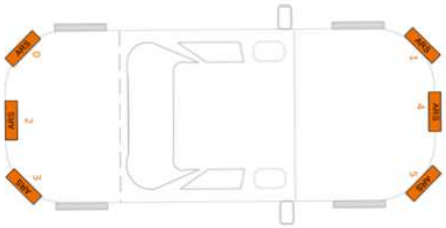
Figura 3 – Sistema de Coordenadas utilizado no *DemoCar*



O eixo X sempre segue o sentido de deslocamento do carro e tem como sua origem o eixo dianteiro do carro. O eixo Y representa os deslocamentos laterais com relação ao carro e tem em sua origem o eixo central do veículo. O eixo Z representa a altura com relação ao chão. O ângulo Yaw é o ângulo que varia quando o carro está virando.

Os radares foram então instalados ao DemoCar como o esquemático da Figura 4. Os ângulos Yaw de cada radar deveriam ser idealmente 0°, 70°, 110°,180°, 250°e 290°.

Figura 4 – Distribuição dos radares no *DemoCar*

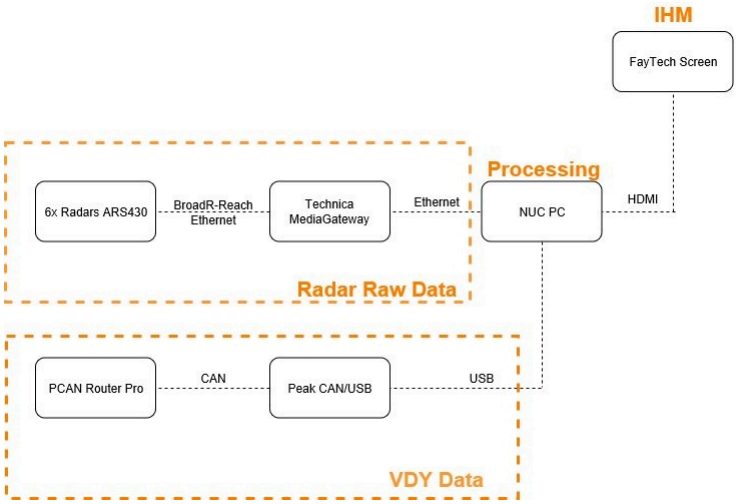


3.3 ARQUITETURA

A arquitetura proposta para o sistema é mostrada na Figura 5.

Cada **radar** se comunica através de um par de canais *full-duplex* com comunicação *BroadR-Reach* [2] limitada a 100 Mbits/s. Para se comunicar com os radares, foi projetado a utilização de um **media gateway**, que é capaz de receber doze pares BroadR-Reach no total, por via de três diferentes switchs. Este aparelho tem a função, então, de receber os dados de todos os seis radares via BroadR-Reach e passar todos os dados juntos em uma só via através de um cabo Ethernet clássico (RJ-45), que tem limitação de 1 Gbit/s. Este cabo

Figura 5 – Esquemático da arquitetura do sistema



é interligado a uma unidade de processamento embarcada, um **Intel NUC**, que receberá os dados dos radares e os enviará como entrada dos algoritmos de alto nível citados na Seção 2.1.

Esses algoritmos precisam ainda de uma outra entrada além dos dados dos radares. Para que eles funcionem, é necessário passar um conjunto de dados fornecidos pelo sistema interno do veículo. Estes dados são bem conhecidos como Vehicle Dynamics (VDY), e no nosso caso se trata da velocidade do veículo, aceleração longitudinal, aceleração lateral e taxa de variação do ângulo Yaw.

Esses dados são fornecidos pelo barramento CAN (*Controller Area Network*) do veículo, onde residem todas as informações do sitema interno do veículo. Então é necessário rotear os dados do barramento CAN do veículo até a nossa unidade de processamento e para fazer isso, usamos um aparelho roteador que se chama **PCAN Router**

Pro, junto a um **Peak CAN-USB** que nos permitirá passar dados do formato CAN para o formato USB e assim finalmente alimentar a unidade de processamento com esses dados.

Por último, no final da cadeia se encontra o monitor utilizado, que é *touch-screen* e permite a um cliente qualquer de ver e interagir com os resultados dos algoritmos. Obviamente, este monitor ajudava também aos desenvolvedores da equipe DemoCar a fazer os testes no carro.

3.3.1 Resumo dos blocos mostrados na Figura 5

a) ***ARS430*** **[1]:** Radar de quarta geração da Continental utilizado no projeto.

b) ***MediaGateway*** **[4]:** Dispositivo quer permite a conversão entre Ethernet BroadR-Reach e Ethernet RJ-45, juntando assim os dados dos seis radares que chegam separadamente em apenas um só cabo de saída RJ-45.

c) ***Intel Nuc*** **[3]:** A unidade de processamento embarcada no carro e usada para todo o processamento do sistema. Pertence a uma linha de mini computadores da Intel e possui Ubuntu 16.04 LTS como sistema operacional instalado.

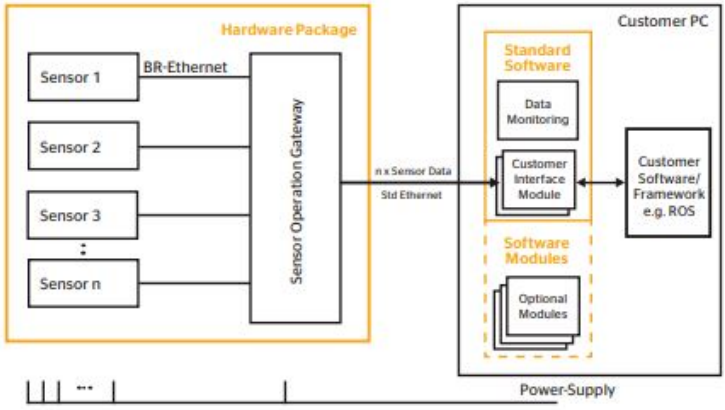
d) ***PCAN Router Pro*** **[5]:** O roteador permite, além de simplesmente rotear os dados do CAN bus do veículo, fazer conversões internas entre os sinais a baixo nível.

e) ***PCAN-USB*** **[6]:** O Peak CAN-USB é simplesmente um adaptador que permite passar do bus CAN ao USB.

3.4 VISÃO GLOBAL DO SISTEMA

A parte hardware do esquemático da Figura 6 mostra os radares se comunicando ao sistema através da MediaGateway. Os módulos de software do sistema servem como uma interface, recuperando os dados dos radares e permitindo aos potenciais clientes de recuperarem estes dados facilmente através do ROS (*Robot Operating System*) ou de uma biblioteca customizada e desenvolver seus próprios produtos com novas ideias.

Figura 6 – Esquemático dos diferentes módulos do sistema e do fluxo de dados



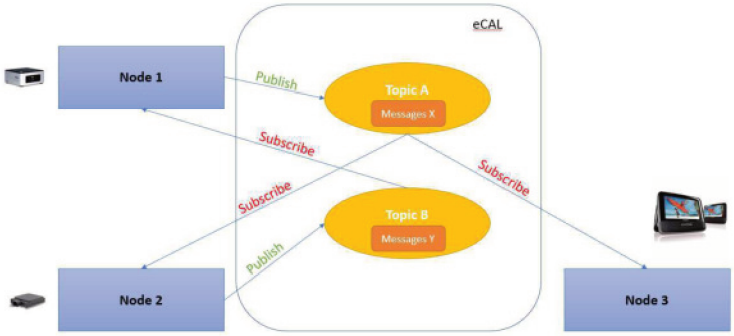
Os algoritmos de alto nível já mencionados neste relatório servem de exemplo para isso. Toda a comunicação intermodular é feita através de um protocolo de transação de dados desenvolvido pela própria Continental que se chama enhanced Communication Abstraction Layer (eCAL).

3.5 ENHANCED COMMUNICATION ABSTRACTION LAYER

O eCAL é um middleware desenvolvido pela Continental que permite alta escalabilidade e performance em transferência de dados. Como já mencionado, seu protocolo de comunicação se assemelha bastante ao *ROS* [7].

Ele foi projetado para um processo de trocas de informação entre diferentes nós. Cada dispositivo ligado a rede contém naturalmente entradas e saídas, logo cada dispositivo se inscreve nos tópicos que contém os dados requerido a ele (entradas) e simultaneamente também publica outros dados (saídas) em um outro tópico, como em um sistema de *cloud*. Os dados são normalmente chamados de mensagens, que contém um padrão pré-estabelecido de envio.

Figura 7 – Exemplo de uma rede eCAL



Este middleware ajuda muito os desenvolvedores em suas tarefas cotidianas, pois ainda contém ferramentas que facilitam monitorar em tempo real, registrar ou reproduzir dados que fluem em um tópico. E mais importante, ele permite uma arquitetura muito mais escalável

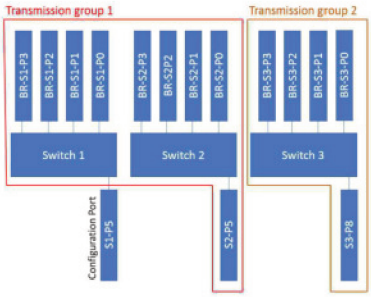
em um ambiente com muitos dispositivos interconectados na mesma rede, pois os dados só fluem quando requeridos por algum dispositivo, e isso diminui em muito a densidade de envio de dados em uma rede.

3.6 CONFIGURAÇÃO DOS EQUIPAMENTOS

3.6.1 Media Gateway

O Technica MediaGateway permite aos usuários um grau de liberdade bem alto para configurar os seus *switch*. Há a possibilidade de permitir ou bloquear o sinal entre diferentes *switchs*, colocar ou tirar LAN tags em sinais requeridos, etc.

Figura 8 – Configuração do MediaGateway



Já pensando na possibilidade futura de conectar também câmeras e no modo de gerenciamento destes dispositivos dentro da media gateway, foi feito uma configuração em que, dos três *switchs* presentes, dois seriam destinados aos radares e o restante seria de uso exclusivo de câmeras satélites. Logo, cada *switch* dentre os dois primeiros, foi conectado a três radares. Esta configuração pode ser vista na Figura 8.

3.6.2 PCAN Router Pro

No bus CAN do veículo, os dados se apresentam codificados pelo fabricante e para que se consiga decodificá-los e, consequentemente, manipulá-los, é necessário ter um arquivo do fabricante chamado de DataBase CAN (DBC).

Os dados são dispostos em uma hierarquia onde existem vários sinais e mensagens, sendo que cada mensagem deve conter um ou mais sinais. Aqui reside todo tipo de sinais elétricos do sistema interno do carro, como velocidade angular de uma determinada roda, o quanto o volante está virado para a direita, se o *air-bag* do motorista está ou não acionado, se os faróis dianteiros estão ligados, etc.

Para acessar esses sinais, começamos por procurá-los por seu ID hexadecimal da mensagem e depois o nome do sinal requerido. Além de simplesmente roteá-los, podemos também manipulá-los como se por exemplo, precisássemos de uma velocidade em m/s e o sinal fornecido se encontra em km/h.

Toda a configuração pode ser feita pelo próprio software disponibilizado pelo fabricante do roteador, chamado PPCAN-Editor 2, ou programado em C e compilado diretamente na memória do roteador. Esta última foi a forma de configuração escolhida e os sinais manipulados eram transmitidos a um período de 2ms. Os sinais transmitidos, que contém os conhecidos dados VDY, foram os seguintes :

a) Velocidade do veículo $[km/h]$;

b) Aceleração transversal $[g]$;

c) Aceleração longitudinal $[m/s^2]$;

d) Taxa de variação do ângulo Yaw $[°]$

3.7 CONFIGURAÇÕES DE SOFTWARE

3.7.1 Sincronização de Tempo (PTP)

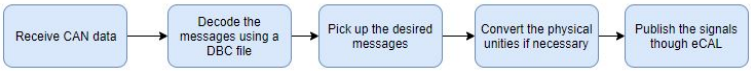
Para que todo o processamento de dados funcione da forma devida, é imprescindível ao sistema conhecer o tempo exato de cada amostra dos dados de radar que são processados. Para isso, o sistema inteiro tem de estar sincronizado, ou seja num determinado momento, todos os dispositivos do sistema possuem o mesmo *timestamp*.

Então, foi implementado um protocolo de sincronização que se chama ***P**recision **T**ime **P**rotocol* (PTP), onde a unidade de processamento serve como *master*, determinando seu clock como a referência principal do sistema e todos os demais dispositivos são *slaves*. Este protocolo foi desenvolvido com auxílio de um programa *open-source* chamado PTPdaemon.

3.7.2 Publicação dos dados VDY na rede eCAL

Como todos os diferentes módulos de software do sistema Cinturão de Radares se comunicam através de eCAL, era necessário disponibilizar também os sinais VDY providos pelo sistema interno do veículo. Para isso, foi desenvolvido um programa escrito em Python que lia continuamente os sinais CAN que chegavam pela porta USB da unidade de processamento e procurava pelo ID hexadecimal dos sinais requeridos. Assim que encontrados, os sinais eram publicados em um tópico específico na rede eCAL e os módulos que estivessem precisando desses dados poderiam vir a se inscrever neste tópico para recuperá-los. O diagrama de fluxo do programa pode ser visto na Figura 9.

Figura 9 – Diagrama de fluxo do programa encarregado de publicar os dados VDY por eCAL.



3.8 PROBLEMA E SOLUÇÃO DE DESALINHAMENTO DOS RADARES

Na Seção 3.2, foi mencionado as especificações de montagem dos seis radares que compõem o Cinturão de Radares, como posição e direção. Como a montagem real sempre é um pouco diferente das especificações ideais e varia de carro pra carro, foi criado um arquivo de configuração no sistema, que permite ao usuário facilmente entrar com as posições e direções reais dos radares. Assim, o sistema é capaz de ler esse arquivo e publicar também os dados por eCAL de maneira contínua, para que o dispositivo que precisar desses dados, possa recuperá-los a qualquer momento.

Acontece que medir exatamente o ângulo Yaw de cada radar é uma tarefa muito difícil na prática e na maioria das vezes os dados entrados pelo usuário não condizem com os dados reais. Sendo assim, é necessário corrigi-los com iterações, alternando sempre entre entrar com os dados iniciais, analisar os resultados, e modificar devidamente os sinais de entrada sobre as direções dos radares.

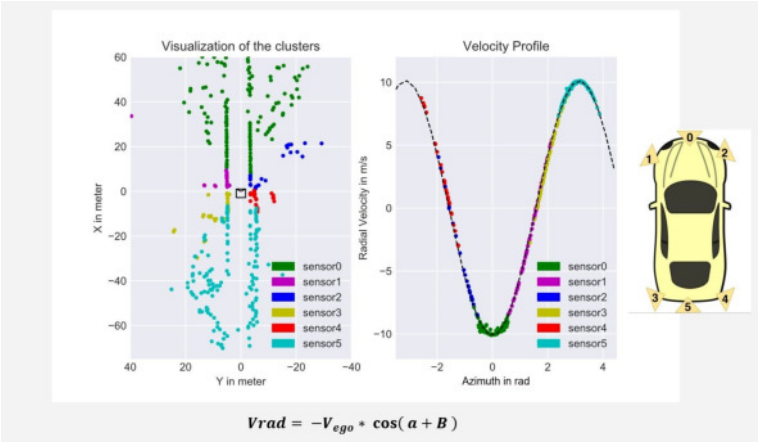
Quando se trata de detecção de objetos estáticos, sabemos que esses objetos possuem velocidade nula. Logo, devido à esta constatação chegamos na Equação 1, onde V_{rad} é a velocidade radial do objeto detectado pelo radar, V_{ego} é a velocidade do carro, α é um ângulo Yaw do radar passado pelo arquivo de configuração e β é um ângulo

relativo do objeto com relação à direção do radar.

$$V_{rad} = -V_{ego} * cos(\alpha + \beta) \tag{1}$$

Logo, podemos concluir que ao plotar a velocidade radial dos objetos detectados, todos os objetos que são estáticos devem estar obrigatoriamente sobre a curva cosseno descrita pela Equação 1, e é em cima desta constatação que é feita a análise e subsequentemente calibração dos ângulos Yaw de cada radar. Para se entender melhor a equação e a teoria apresentada, podemos ver a localização de objetos estáticos de cada radar sob a curva cosseno azimutal na Figura 10.

Figura 10 – Objetos estáticos detectados pelo Cinturão de Radares e o plot de suas respectivas velocidades radiais

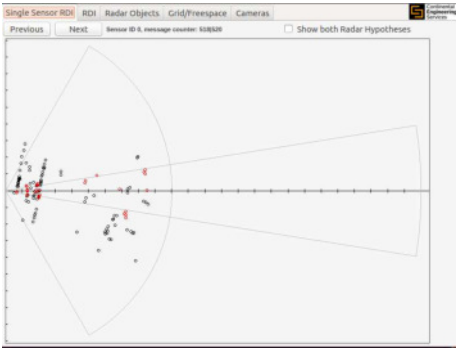


4 RESULTADOS

Para começar esta seção, segue abaixo imagens de cada aba da ferramenta de visualização do Cinturão de Radares :

a) **Detecções de um único radar:**
As detecções de um único radar são mostradas na Figura 11. Os pontos vermelhos são detecções advindas do scan de alta distância, e os pontos pretos detecções advindas do scan de curta distância. Nós podemos pela ferramenta escolher qual dos radares queremos visualizar.

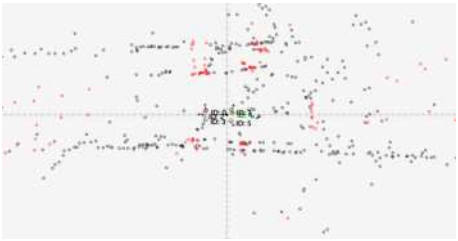
Figura 11 – Objetos detectados por um único radar



b) **Detecções de todos os radares do cinturão:**
A Figura 12 mostra as detecções "brutas"como na figura anterior, porém esta contém os dados de todos os radares ao mesmo tempo e é como se fosse uma vista de cima do veículo. Nesta imagem podemos perceber o uso do arquivo de configuração que passa as posições e direções de cada radar. O sistema faz uso da posição para posicionar os radares

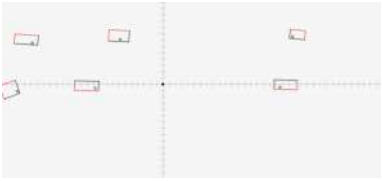
em seus devidos lugares no mapa e então indica a direção (ângulo yaw) com as flechas, as detecções de cada radar são enfim mostradas ao interior do cone em frente ao radar.

Figura 12 – Vista de cima de todos os objetos detectados por todos os radares do cinturão



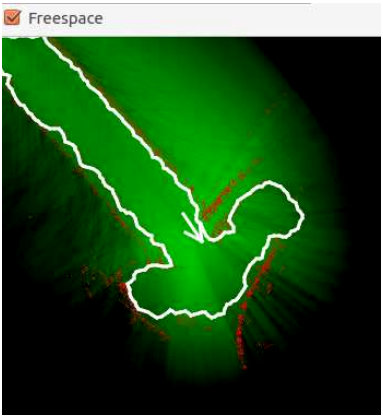
c) **Rastreador de Objetos Dinâmicos:**
A aba de visualização do algoritmo Rastreador de Objetos Dinâmicos mostra uma caixa para cada objeto dinâmico detectado pelo Cinturão de Radares. O mapa segue o mesmo sistema de coordenadas AUTOSAR mencionado na Seção 3.2 para falar sobre as posições relativas ao carro.

Figura 13 – Rastreador de Objetos Dinâmicos



d) **Mapa de Ocupação e Detector de Espaço Livre:** Esta última aba mostra dois algoritmos juntos que se sobrepõem. O Mapa de Ocupação cria um mapa verde ao redor do veículo e mostra até onde o sistema foi capaz de captar informação e também mostra todas as detecções de objetos estáticas em vermelho. Assim, o Detector de Espaço Livre usa essa informação para calcular a área livre que determina onde o carro pode passar, esta área sendo mostrada em branco. No exemplo da Figura 14, podemos ver o veículo chegando em uma rotatória.

Figura 14 – Mapa de Ocupação - Detector de Espaço Livre



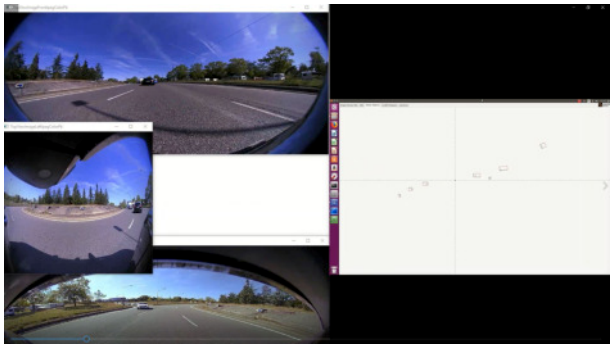
Em seguida, será apresentado mais explicações sobre esses dois últimos algoritmos de alto nível que de certa forma, servem de exemplo aos potenciais clientes para mostrar o grande potencial de performance dos dados advindos de radares em um veículo. Ambos os algoritmos precisam obrigatoriamente dos dados VDY e da posição/direção de cada radar. Eles recuperam esses dados, assim como as detecções dos

radares, através de tópicos eCAL.

4.1 RASTREADOR DE OBJETOS DINÂMICOS

Este algoritmo visa a estimar a posição e o tamanho de objetos dinâmicos ao redor do veículo.

Figura 15 – Rastreador de Objetos Dinâmicos detectando múltiplos carros em sua volta.



Depois de muitos testes em diversos ambientes, foi possível obter resultados bem satisfatórios com esse algoritmo. Sua estimação sobre o tamanho dos objetos se mostrou confiável, tornando possível diferenciar carros de caminhões ou de ciclistas e pedestres. Pelos testes, foi possível comprovar que o sistema não possui pontos-cegos e um objeto pode ser bem detectado até uma distância máxima de 150m.

O problema apresentado na Seção 3.8 era possível de ser identificado visualizando esse algoritmo. Os objetos estáticos que se encontram fora da curva cosseno azimutal acabavam por serem considerados objetos dinâmicos e eram mostrados na janela de visualização como

objetos-fantasma (que não existem). Eles piscavam bastante e não se moviam, mas após a calibração do sistema, o problema foi resolvido.

4.2 MAPA DE OCUPAÇÃO E DETECTOR DE ESPAÇO LIVRE

Para a representação do ambiente estático em volta do veículo, o Mapa de Ocupação começa por mapear todos os objetos estáticos detectados individualmente por cada radar. A acumulação de todas as detecções com o passar do tempo vão sendo consideradas mais confiáveis e só a partir de determinado limite de confiança são plotadas. O algoritmo usa o mapa atual de detecções estáticas e os sinais VDY do veículo para se localizar espacialmente e locomover o veículo sobre o mapa.

Após ter disponível o mapa de objetos estáticos advindo do Mapa de Ocupação, o Detector de Espaço Livre tenta extrair a área em volta do veículo que não está obstruída. Seu resultado então é uma fronteira, onde é considerado que o veículo pode se locomover sem problemas.

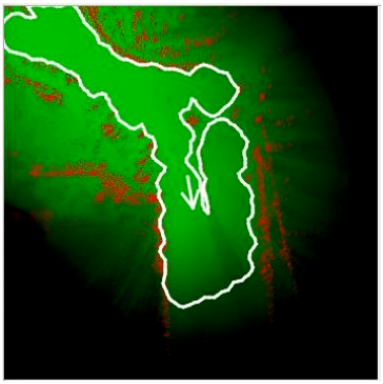
Na maioria dos casos, os algoritmos se comportaram muito bem. Nos casos mais fáceis, como rodovias, ou ruas simples de mão única ou dupla, o algoritmo consegue extrair perfeitamente a área de sua rota. No caso de rotatórias, ele se comportou bem também, sendo possível ver o caminho da rotatória, assim como todas as suas saídas.

Porém, tiveram alguns casos específicos onde o algoritmo ainda se mostrava com muita dificuldade de calcular o seu espaço livre, como bifurcações, estacionamentos e ruas com canteiros centrais. Nas bifurcações, ele demora a perceber que existe uma barreira entre ambas as rotas que se bifurcaram, indicando que ainda há possibilidade de trocar de rota; em estacionamentos, ele tem dificuldades em considerar carros parados como obstáculos e acaba se perdendo, assim é possível ver os limites das zonas livres, identificados pelo polígono branco na

Figura 17, desaparecer durante alguns *frames* nesse tipo de ambiente; e finalmente em ruas com canteiros centrais, ele tem dificuldades em captar o canteiro como um obstáculo e normalmente apresenta como resultado os dois lados da via como espaços livres.

Figura 16 – Imagens de câmeras satélites utilizadas para analisar visualmente o ambiente que está a ser analisado pelos algoritmos usando sinais dos radares.

Figura 17 – Visualização dos algoritmos Mapa de Ocupação e Detector de Espaço Livre.



5 CONCLUSÃO

O trabalho de integração do sistema do Cinturão de Radares em um veículo real foi cumprido com sucesso. Vários testes foram feitos para se avaliar a robustez do sistema, e ele passou em todos, rodando de maneira estável e contínua. Ao final, o sistema foi deixado pronto para ser ativado ao lançar-se um único script que se encarregaria de fazer todas as tarefas necessárias para o funcionamento do sistema de maneira completa e no futuro é desejável de que o sistema se lance automaticamente, assim que o carro é ligado.

O projeto DemoCar ainda se encontra em seu primeiro estágio, que é de obter a informação do ambiente ao redor do veículo por meio de vários diferentes tipos de sensores. Nos próximos estágios, o projeto começará a fusionar dados de diferentes sensores para aumentar a confiabilidade dos dados, diminuir a complexidade de algoritmos existentes e fazer o veículo finalmente ser capaz de tomar decisões por si só em casos cotidianos do trânsito.

Agora que a integração do Cinturão de Radares está completa e rodando no DemoCar, já é plausível imaginar a fusão de sinais dos radares com as imagens de câmera por exemplo. Como imagens advindas de câmeras são normalmente arquivos pesados, com alta densidade de informação, os algoritmos de processamento de imagens são muito lentos, tornando impossível o seu uso em funções de tempo real. O radar, que pelo contrário contém dados bem leves, pode fazer as detecções primeiramente, reduzindo as dimensões das imagens de câmeras a serem analisadas e, consequentemente, diminuindo a complexidade dos algoritmos. Uma interessante tabela que compara várias características de diferentes sensores se encontra na Figura 18 e com ela podemos ter uma boa ideia de como a fusão de dados nos ajudará no futuro.

Figura 18 – Tabela comparando diversas características entre três diferentes tipos de sensores.

COMPUTER PERCEPTION ANALYSIS

	Camera	RADAR	LiDAR
Object Detection	Medium	High	High
Classification	High	Medium	Low
Density of Raw Data	High	Medium	High
Velocity Measurement	Low	High	Low
Lane Detection	High	Low	Low
Sign Recognition	High	Low	Low
Rain, Fog, Snow Vision	Low	High	Low
Night Vision	Low	High	High
Cost	Medium	Low	High

Acredito que o radar tem uma forte contribuição neste quesito de fusão de dados, pois possui características únicas que podem ajudar em muitas situações diferentes no trânsito. Se confirmado, o Cinturão de Radares será o ponto de partido de algoritmos muito interessantes em um futuro próximo, e o projeto DemoCar estará cada vez mais perto de seu objetivo final que é obter um carro autônomo.

REFERÊNCIAS

[1] *ARS430. Advanced Radar Sensor*. Continental. Disp. em: https://conti-engineering.com/components/ars430/.

[2] *BroadR-Reach. Enabling One Pair Ethernet*. Broadcom Corporation. 2012. Disp. em: http://www.ethercat.org/2013/mobile_applications/files/04_EtherCAT_Mobile_App_Broadcom.pdf.

[3] *Intel Nuc. D54250WYK*. Intel. 2016. Disp. em: https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/nuc-kit-d54250wyk-product-brief.pdf.

[4] *MediaGateway*. Technica Engineering. Mar. de 2018. Disp. em: https://technica-engineering.de/wp-content/uploads/2018/06/MediaGateway-UserManual-V4.2.98-1.pdf.

[5] *PCAN Router Pro*. Peak System. 2019. Disp. em: https://www.peak-system.com/produktcd/Pdf/English/PCAN-Router-Pro_UserMan_eng.pdf.

[6] *PCAN-USB*. Peak System. 2019. Disp. em: https://www.peak-system.com/produktcd/Pdf/English/PCAN-USB_UserMan_eng.pdf.

[7] *ROS. Robotic Operation System*. Open-Source project. 2020. Disp. em: https://www.ros.org/about-ros/.

ANEXO A – RELATÓRIO ORIGINAL

Logo em seguida, como anexo, é apresentado o relatório original, escrito em inglês, do estágio de seis meses realizado para a conclusão do curso de engenharia eletrônica na instituição de ensino superior ENSEEIHT - INP, Toulouse.

Electronics and Signal Processing

Internship Report

Radar Belt System integration and Mirror View project

Student:
Vitor de Castro Carvalho

Supervisors:
M. Cedric Dulau
M. Jacques Belloc

Date of submission
September 12, 2019

Abstract

Over the last decade, several statistics studies were done to find the main causes of the car accidents over the world. The results show clearly that the main causes are always related to the driver. The problems can be due to perception, interpretation, evaluation, decision and, finally, action of the driver. Normally, the driver always misses some information of the environment around the car, aiming him to take bad decisions.

On this context, it was created a new area of expertise on driver solutions called ADAS (Advanced Driver Assistance Systems), which are intelligent systems that reside inside the vehicle and assist the main driver in a variety of ways. These systems are responsible to reduce vehicular accidents and fatalities, giving the driver some information about the environment with the assistance of many integrated sensors on the car. Furthermore, this assistance can become an active safety system, taking control over the vehicle's functions in specific cases.

On this perspective, I put my efforts over the last six months to develop my final internship called *"Integration of environment perception and localization systems"*. The two main goals of this internship were :

1. *Radar Belt :* Do the hardware and software integration of a system, which consist in merging the data of six radars to do a centralized object tracking and to detect the free space around the car.
2. *Mirror View :* Develop an application to replace the rear mirrors of the car, eliminating blind spots and integrate the hardware.

Therefore, this report will give information about the company in which I worked for, its client, a more detailed context about the internship's subject and the development of the two respective projects above.

Contents

List of Figures

1 Company Presentation

1.1 Celad

The company in which I worked is called Celad society and it is a consulting enterprise created on the year 1990 in Toulouse. With eight others company's branches, Celad counts with more than 1100 workers and more than 200 different clients.

Since it was created, Celad keeps getting a regular growth year by year, reaching a good financial health and making it receive a note of excellence "cotation Banque de France $C3++$". This excellent growth is only reached due to the focus on certain competences of the engineers and consultants as : team service, re-activity, professionalism and implication.

Figure 1.1: Celad's logo

As its main activity, Celad purposes to help its clients on the conception and development of solutions on several projects of all kinds. This help is achieved by two different manners, the first one is when the consultant works directly on the client's local, which represents the big majority, and the second one regards the engineers who work at Celad's local.

Finally, in my case, the internship took place in the client's office at Continental Toulouse.

1.2 Continental

Founded in Hanover in 1871, Continental AG society, mostly known by its tires, is a German automotive manufacturing company specialized in brake systems, interior electronics, automotive safety, tires and other parts for the automotive and transportation industries. This diversification has started in 2007, when Continental has decided to change its strategy and diversify the markets.

Nowadays, the company has grown hugely and it is present in 60 countries around the five continents with 544 offices at total. It's divided on five major parts, being them *Chassis and Safety* whose projects concern mounting sensors around the car to know better the environment

around it and take care of the passengers; *Powertrain* which develops innovative motors to consume less energy, taking care about the ecologic side at the development of a car; *Interior* which is responsible for the Human-Machine Interaction inside the car and connectivity among all the sensors; *Tires* which works on the development and production of the tires, trying to improve performance with lower costs on an ecologic way. The last one is called *ContiTech* and it develops and produces functional elements and systems for the vehicle construction and for some other industries as well.

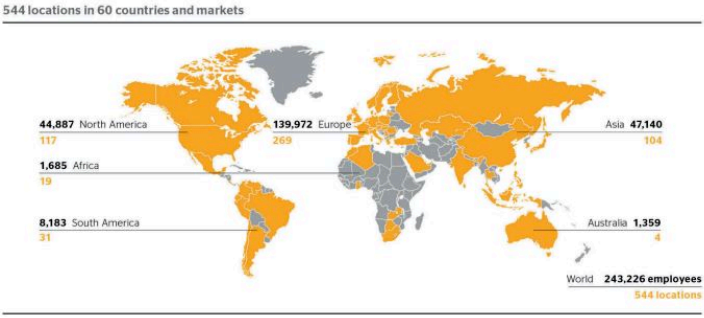


Figure 1.2: Continental's locations around the world

1.2.1 The CES France group

My internship was developed in the CES (**C**ontinental **E**ngineering **S**ystems) group, that belongs to the Chassis & Safety division.

The CES is present on 20 locations in the focal regions of Europe, North America, China and Japan with more than 1700 employees. It works on selling its services among Continental's divisions and to external companies as well, providing development teams and specialized components (camera, radar, LIDAR, GPS, etc.) to achieve the project's goals.

In Toulouse, there are around 150 engineers specialized in signal processing, image processing and Human-Machine Interaction (HMI) working for the CES group. Among all the projects, they are divided mainly in projects concerning fusion data for driver assistance, and the development of new internal products.

Its areas of expertise are : Driver Assistance, Brake Systems, Interior Electronic Functions, Drive-line/Electrification and Product Solutions.

1.2.2 Advanced Driver-Assistance Systems (ADAS)

One of the main activities in CES consists in developing, with the *Chassis and Safety* division and others external clients, driving functions to assist the driver in several occasions with the

information of the sensors around the vehicle. Moreover, ADAS systems can take the control over some vehicle's actions, which is called "active assistance".

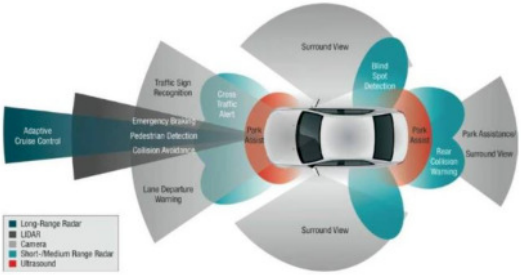


Figure 1.3: ADAS functions and sensors

ADAS are electronic systems born to help the vehicle driver while driving, but nowadays they are fully automating the driving tasks inside the car. This area is where the internship was developed, integrating new features on the Continental's Demonstration Car (DemoCar), firstly to assist the driver and then fully automate the car, making it autonomous.

There are five levels of an automated driving, beginning with low assistance jobs as parking assistance and speed control. After, it comes the partial automation, which turns the vehicle allowed to perform brake, steering and acceleration, but the human still monitors all tasks and can take control at any time. On the third level, Conditional Automation, the vehicle can perform most driving tasks as it detects a lot the environment around it, but human override is still required. Then, it comes High Automation, where the vehicle performs all the driving tasks under several circumstances but human override is still an option. Finally, Full Automation demands zero human attention or interaction making the car completely autonomous.

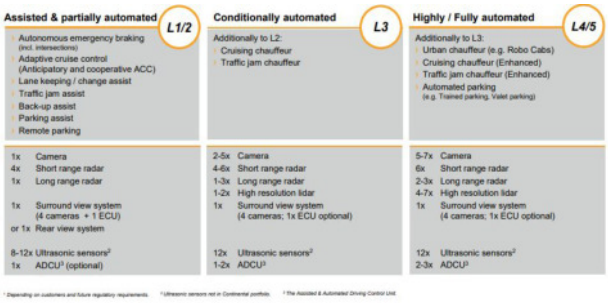


Figure 1.4: Automation levels of a vehicle

Among all the projects of the ADAS group, we can mention for example the project Emer-

gency Brake Assist (**EBA**), the project Traffic Sign Assist (**TSA**) or the project lane Departure Warning (**LDW**), whose goals are described below :

- The **EBA** project, based on a radar system, is an urgency brake assistant that allows the vehicle to brake itself if it too close of another vehicle, in front of it, on a dangerous speed.
- The goal of the **TSA** project is to remind the driver about the current limit speed of the road. This technology uses a camera and image processing to distinguish the limit speed signals on the road.
- The **LDW** project emits a warning to the driver while the vehicle is crossing the line that quits the road. Regarding this subject, there is still one other example, the Lane Keeping System (**LKS**) which this time is an "active assistance" that slightly turns the steering wheel while crossing the line to give more time to the driver's reaction.

These systems operate with the knowledge of the environment around the car, which requires the integration of many sensors. As the automated level increases, the number of required sensors also increases, but their performance overpass a lot the human being, once they can detect the objects in higher distances, they don't need luminosity and they can detect on a 360° view.

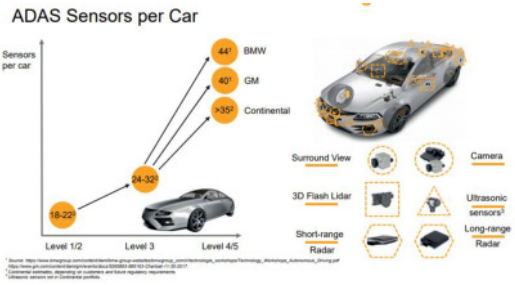


Figure 1.5: Number of sensors per car

1.2.3 Demonstration Car (DemoCar) project

The DemoCar project has commercial interests inside CES, as it's an excellent showcase for CES to show to its clients what is being developed internally. It was inside this project that the internship was developed. Currently, there are eight engineers working on this project.

Running parallel integration on the car, the team works with a VeloDyne LIDAR, four cameras aiming to have a 360° view around the car, two cameras to replace the vehicle's rear view mirrors and six radars, so the tasks concern the software adaptability, architecture hardware conception, hardware integration and also data fusion development.

The goal is to have working on the car the most recent technology and algorithms developed by the CES group. There are four steps in the the development of this project. We are currently on the first of the steps described below :

Figure 1.6: The DemoCar from CES Toulouse

1. DriveSense : Integration of all the ADAS technologies that do not take control over the vehicle. So, these are the technologies responsible to get the environment around the car.
2. Integration of new technologies that prepares the lateral and longitudinal control over the vehicle. This includes data fusion between different sensors.
3. Vehicle partially automated with collaborative driving between human being and vehicle.
4. Autonomous control of the vehicle driving at low speeds and inside a controlled environment.

1.2.4 Agile Method

The DemoCar project is organized following an Agile software development method called Scrum. This method aims to have an adaptive planning, evolutionary development, early delivery and it also encourages rapid and flexible response to change from teams.

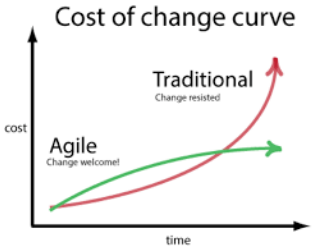


Figure 1.7: Cost of change comparison

It is designed on a certain way that teams break their whole work into little tasks, which have to be completed within timeboxed iterations, called *sprints*, during two weeks. Each sprint has

a planning meeting to define the tasks, the priorities and the validation criteria for each task. It has also a demo meeting, where the stories should be validated following a determined criteria and the members may show their sprint results. Each of these meetings should take no longer than one hour.

Moreover, everyday the team does a daily meeting in which every member should says what he has done on the last day, what he is going to do on the current day and mainly focus on the problems he has faced anf if some help is needed. So anybody else who can help him to solve the problem might talk. This daily meetings are called *stand-ups* and should take no longer than fifteen minutes.

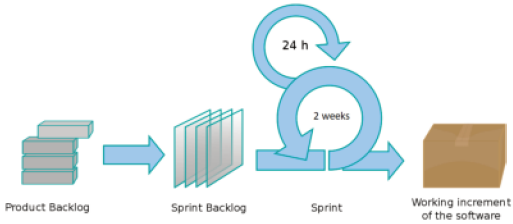


Figure 1.8: Scrum workflow

This method proved itself to be very good as it allows : ease integration as the work flows, to catch the issues early, to build a reusable base and to improve task sharing.

2 The internship's context

The projects concerned by this internship are called **Radar Belt System** and **Mirror View**. They are sub-projects developed inside the DemoCar project with the goal to extract some information about the environment around the vehicle.

2.1 Radar Belt System Integration

2.1.1 Presentation

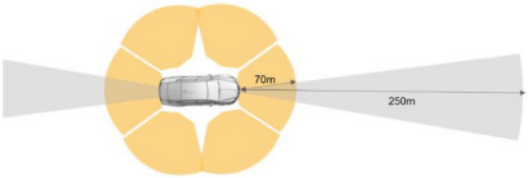


Figure 2.1: 360° perception with 6 ARS430 radar sensors

The Radar Belt System integration is the first of two parallel projects developed during this internship. The Radar Belt System is a multiple sensor prototype system built to have a 360° radar perception around the vehicle. The "belt" consists of six Continental's fourth generation radars (ARS430) around the car allowing a 360° perception on short range.

Each ARS430 radar runs two scans per sample, being the far scan and the near scan maximum range 250m and 70m respectively. The far scan has narrower field of view so it can cover longer distances, then the 360° view is only possible with the near scan. The ARS430 field of view is shown on the figure 2.2: the **N**ear **R**ange **S**can (**NRS**) is yellow and covers 150° up to 10m and 90° up to 70m; the **F**ar **S**can **R**ange (**FRS**) is shown in grey color and covers 18° up to 250m. Both scans are done at the same cycle, being **one cycle time** equals to 60ms.

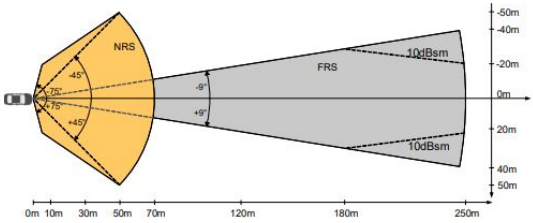


Figure 2.2: ARS430 Field of View

These radars have an internal processing that outputs a list of raw detected objects, so this system allows the user to get all these objects from all radars and do some post-processing to

develop additional algorithms. The list contains several informations for every single detection. These outputs are listed below and better explained as an illustration on the figure 2.3.

- **Radial distance [m]**
- **Radial relative velocity [m/s]**
- **Probability of ambiguity [%]**
- **Azimuth angle [°]**
- **R**adar **C**ross **S**ection **(RCS)**

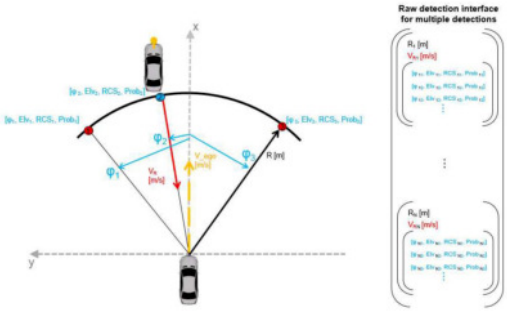


Figure 2.3: ARS430 output

The additional algorithms to be integrated on the car were the **Centralized Object Tracking**, that merges all the raw detected objects from each radar and compares their velocity with the vehicle's velocity, so it can highlight the dynamics objects around the car. This feature is very important to detect alive objects as cars, bikes, pedestrians, etc. The **Occupancy Grid** that merges the raw data to find the statics objects around the car and the **Free Space** that uses the Occupancy Grid information to calculate the space where the car can move without further problems, really essential for the autonomous driving.

The low level communication of the system works independently. However, for the additional algorithms described above to work, they need some extra information about the ego motion of the vehicle as its velocity, longitudinal acceleration, lateral acceleration and yaw rate. These data are already available inside the vehicle's internal system and it might be recovery to pass to the algorithms the information they need. These kind of data is mostly known as Vehicle Dynamics (**VDY**).

2.1.2 Goals

The core of the Radar Belt software was fully developed by CES Frankfurt. Therefore, the tasks concerning the internship were mostly about integration on the car.

On this way, the goals previewed for the project were :

1. Determine a hardware architecture to integrate the system on the car;
2. Configure the switch responsible for the communication between radars and the processing unit;
3. Configure the processing unit which should be embedded on the car;
4. Study the Continental's internal protocol of communication;
5. Configure the PCAN Router responsible to get the VDY data from the internal CAN bus of the vehicle;
6. Connect all the elements on the chain and run it;
7. Do some recordings, analyze and validate the results.

2.2 Mirror View project

2.2.1 Presentation

The second project concerned by this internship is the Mirror View. It is a project intended to replace the physical rear view mirrors by a set of cameras and screens, so it would not be necessary to have physical mirrors outside the car anymore. Instead of it, we can place cameras which might take much less space outside the car and the screens rest inside the car at an ergonomic position to the driver.

Figure 2.4: A practical example of the Mirror View project designed by Audi

Barely every mirror has distortion effects, normally at the borders where it is slightly curved. These distortion effects were one of the main points for this project, where the idea was to establish a likelihood between the screen image and the mirror image, so the drivers could familiarize themselves faster with this technology. For this reason, experimental purposes were purposed to simulate some different kinds of distortion.

Cameras also introduce distortions to the images they acquire from the real world. As a consequence, some calibration has is necessary, which is done with a 3D image of an object with a very known form, like squares, so we can map how they should be in 2D and there calculate the parameters of of the distortion model presented. These parameters are called intrinsic parameters, when introduced by the set of lens, and extrinsic parameters when related to the camera emplacement in the world perspective.

Therefore, the central idea of this project is to do the acquisition of the images of the cameras placed outside the car, do a calibration on them to eliminate intrinsic and extrinsic distortions, apply a distortion at the border of the images and display the resulted images on the screens.

There are two main advantages in the center of this project. The first one and most obvious is that we will not need the mirrors outside the car anymore, this might be good for designers to get a more beautiful vehicle and it is also good for the vehicle's aerodynamics at high speed, so it would save some fuel.

The second advantage is more related to the merge of all the vehicle's sensors functions. Even if we had a mirror without blind spots, the fact that the image is displayed on a screen instead of a mirror, gives us a lot of opportunities to overlay some features on it. So, in a near future, it should be possible to highlight detected objects on these images or others features.

Finally, this project was previewed to run inside the Continental's internal framework, as it is a very powerful tool to do some necessary actions, such as the acquisition and the display. It has already included into it the Continental's cameras modules allowing the communication with the cameras that are already embedded in the car.

2.2.2 Goals

As this project is still at the beginning phase, the goals for this internship concerned only the basic features, which are doing the acquisition, calibration, applying the distortion and finally displaying the images on the screen. As the very last phase of the internship is the integration of the project in the car, it would involve the planning of the hardware architecture to embed the project.

To start the development of this application, it was used a Continental's **framework** which aims to facilitate exactly the creation of applications that handles data acquisition from sensors and post-processing.

Therefore, the previewed goals for the internship were :

1. Familiarize with the internal framework;
2. Create an application for the Mirror View project inside the framework;
3. Code the image processing methods of the application;
4. Test with at least two cameras and analyze if it respects the real-time constraints of the application;
5. Create the architecture to embed the project in the car;
6. Integrate the Mirror View in the car.

2.3 Planning

Radar Belt integration/ Mirror View			DURATION (days)
START DATE	END DATE	DESCRIPTION	
3/4/19	3/11/19	Adaption/Planning	7
3/11/19	4/11/19	Hardware Configuration	30
4/11/19	4/19/19	Basic functions integration	8
4/22/19	5/17/19	VDY Publisher	25
5/20/19	5/31/19	PCAN Router Programming	11
6/3/19	6/28/19	Algorithms Integration	25
7/1/19	7/5/19	Create the MV app	4
7/8/19	7/19/19	Structure the code	11
7/22/19	8/2/19	Calibration/Distortion	10
8/5/19	8/9/19	Optimization	4
8/12/19	8/30/19	Integration	18

Figure 2.5: Gantt Diagram for the internship

3 Radar Belt System Integration

3.1 Motivation

The task of making a car autonomous demands the utilization of several different sensors, each one with some advantages and disadvantages. For this reason, merging different sensors data might be very useful, once it could give us more information and decrease the complexity of the algorithms.

Radars are well known for their low cost to the manufacturers. In addition, their density of raw data is also low, decreasing the difficulty of the post-processing algorithms. Moreover, in comparison with all the others available sensors, radars have a really good performance in tough conditions like rain, fog, snow and even at night.

As a result, Radar Belt system caught the eye of many Continental clients and investors as a powerful project. With this simple set of six radars around the car and some relatively post-processing, we might have a powerful tool to detect objects around the car at any condition, since the signals come from electromagnetic waves reflections.

3.2 State of the DemoCar at the beginning

At the very beginning of the internship, the DemoCar had installed four short-range radars and two long-range radars. However, using the fourth generation Continental's radars, it was not necessary to have both anymore, once this new radar is able to do a near scan and a far scan once per sample.

Then, at this time, while I was getting used to the Agile process and the constraints of the project to build the new architecture, there was one person who works on the garage replacing these old radars by the six ARS430.

As mounting parameters obviously matter, the coordinate system used for vehicles is explained briefly on the figure 3.1:

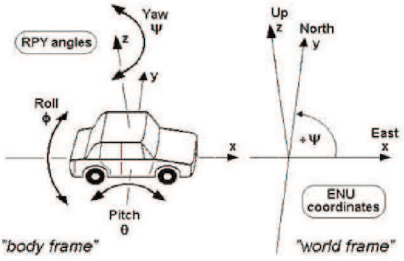


Figure 3.1: Coordinate systems with RPY angles for cars

Therefore, the x-axis is the longitudinal direction of the vehicle, the y-axis is lateral direction, the z-axis is the height regarding the ground, the yaw angle is the most important one, as it is the

angle which changes while the vehicle is changing its direction, the roll angle is the angle around the x-axis, the pitch angle is the angle around the y-axis and these last two angles normally won't change much if the load inside the vehicle does not change as well.

The data-sheet of the ARS430 shows the mounting tolerances for each radar. They are at maximum 600 mm ideally on the y-axis, but it could go up to 900 mm, and at minimum 295 mm from the ground on the z-axis, at maximum 800 mm ideally, then it could go up to 1000 mm if necessary, however some responses might be compromised.

The yaw angle of the four radars at the corners of the vehicle should be ideally 70°, 110°, 250° and 290° respectively. On the figure 3.2 is possible to see how the ideally distribution of the mounted radars around the car.

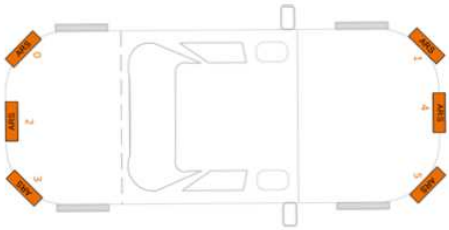


Figure 3.2: Radars distribution around the DemoCar

3.3 Architecture

As the first project of the internship, I started working at the Radar Belt System integration. To begin this work, it was necessary to purpose an architecture for the embedded Radar Belt system, so the necessary equipment could be bought and then configured. Thus, the purposed system's architecture is shown on the figure 3.3.

Each **radar** has a pair of full-duplex twisted cables responsible for the Ethernet BroadR-Reach communication and the data can be transferred at a maximum rate of $100 Mbits/s$. Then, all the radars signals go to a **media gateway**, which has 3 switches, each one being able to deal with 4 pairs BroadR-Reach. The media gateway's output is a Gigabit Ethernet cable with speed up to $1 Gbits/s$ and it goes directly to the chosen processing unit, called **Intel NUC**.

The VDY data might be recovered from the vehicle's CAN bus, where resides all the data flow of the vehicle's internal electric system. We were only concerned about the information inside specific messages : vehicle speed, longitudinal acceleration, lateral acceleration and finally yaw rate. Therefore, we route these data available in the CAN bus to an output which we can connect our system and get all the information needed. Thus, we do this through a **PCAN Router** and also a **Peak CAN/USB**, responsible to convert the interface CAN into an USB connector, so we can connect it to the processing unit.

Lastly, at the end of the chain, it is the **screen**, which allows us to control the processing unit and also see the results. This section is usually called **H**uman-**M**achine **I**nteraction (HMI).

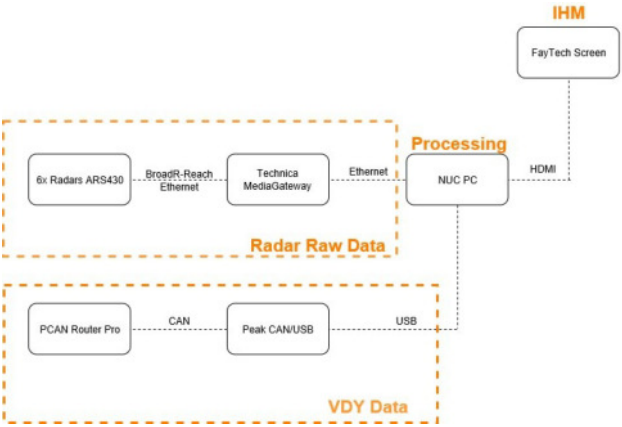


Figure 3.3: General architecture of the Radar Belt system

3.3.1 ARS430

As the core of this project, the ARS430 is a Continental's fourth generation radar with an operational frequency of 77 GHz. As already mentioned, it has a broad field of view realized by two independent scans (near range and far range), being able to detect objects up to 70m of distance at near range and 250 m at far range.

Figure 3.4: The Continental's fourth generation radar ARS430

As interface options, it could communicate via Ethernet BroadR-Reach, CAN or Flexray. And for this project, it was chosen the Ethernet communication that could go up to $100MBit/s$.

Moreover, its output has a format of a list with all the detected objects called Radar Detection

Image (**RDI**). This RDI contains a lot of information for each object detected.

3.3.2 Technica Engineering MediaGateway

To connect the radars with the unit processing of the system, it's necessary some kind of converter to pass from the BroadR-Reach cables to a simple Gigabit Ethernet cable, commonly present at all unit processing devices. For this purpose, we have chosen a MediaGateway.

The MediaGateway from technica Engineering is a development tool for testing and analyzing on-board vehicle networks. Its in-built automotive switches enable both capture of traffic between devices while maintaining the normal communication, as well as interaction with such devices via your test set-up. It supports single and double tagging of VLANs, Mirroring and Forwarding.

Figure 3.5: MediaGateway from Technica Engineering

3.3.3 Intel NUC

The processing unit in this project is responsible to receive radars data, VDY data and consequently do the post-processing with them. The **N**ext **U**nit **C**omputing (**NUC**) is a very small desktop computer replacement that uses the ultra compact form factor. It is built to be a very small PC that fits perfectly with the needs of an embedded computer at size, being simultaneously very powerful.

For this project, it runs Ubuntu 16.04 LTS.

It has:

- 4x USB ports
- 1x mini Display port
- 1x mini HDMI port
- 1x Ethernet port

Figure 3.6: The Intel NUC computer

- 1x Intel processor i7 3.40GHz
- 1x 16 GB RAM

3.3.4 PCAN Router PRO and Peak CAN-USB

Many sensors attached to the car require vehicle data to work. The Radar Belt system does not require it to output the radar raw detections, but it does require it for the proper functioning of its additional algorithms.

Each manufacturer uses its own CAN message encoding settings and, very often, a CAN router is mandatory to make inputs match outputs. Hence, it was necessary to program a CAN router to pick up only the interesting messages (VDY data) for us and route them as an output for further utilization.

The chosen CAN router is called PCAN-Router Pro, from PEAK. It allows to join the data traffic from four High-speed CAN busses. The behavior of the router is configured via the CAN bus with the provided Windows program PPCAN-Editor. As well as pure forwarding, the CAN data can be processed, manipulated, and for example, filtered in a number of different ways. There are a variety of function blocks and other settings available to the user for configuration setup.

As an interface between CAN router and computer, there is usually a converter $CAN \rightarrow USB$, in this case it comes also from PEAK and it's called simply Peak CAN-USB.

Figure 3.7: The PCAN Router Pro

3.4 System Overview

This part aims to give a brief overview about the whole system, including the software side inside the processing unit. There is the hardware package, with the radars connected to the system through a Media Gateway and there is also the software modules such as an interface that allows the customer to develop its own algorithms and communicate easily with the system and the visualization.

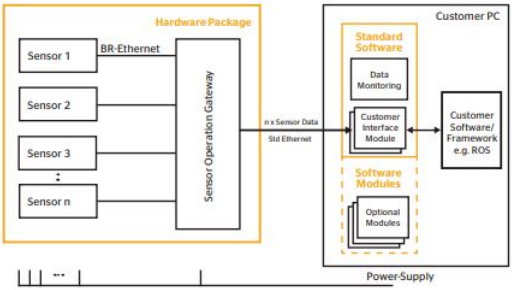


Figure 3.8: System Concept Radar

The additional algorithms already mentioned recover the radars data by the interface module and do their post-processing. The client might want to develop its own algorithms, so it should recover radars data by the same way. All this communication among modules is done by a Continental's internal communication protocol called eCAL.

3.5 enhanced Communication Abstraction Layer (eCAL)

The **e**nhanced **C**ommunication **A**bstraction **L**ayer (eCAL) is a middleware that enables scalable, high performance interprocess communication on a single computer node or between different nodes in a computer network.

The eCAL communication architecture is very similar to that of **R**obot **O**perating **S**ystem, very known as **ROS**, which is a set of several open-source software frameworks to robotics software development wide used in Linux operating systems.

It is designed for typical cloud computing scenarios where different processes exchange their I/O's using a publisher/subscriber pattern just like ROS. The data exchange is based on so called topics. A topic wraps the message that should be exchanged and can be connected to more than one publisher and/or subscriber.

Therefore, firstly a **node**, decided to share some data, start to publish **messages** into a **topic**. Any node interested in these data might subscribe to this topic, thus a connection is done between **publisher** and **subscriber**. The process is very flexible in many ways : the messages are serialized standard data, so it is very fast; any node can be publisher and subscriber of different topics simultaneously; a single topic may have several subscribers simultaneously.

The **callback** is not obligatory, but it might be very useful as it was invented to avoid polling.

- **Node:** A node represents a single process running eCAL. This node always has a unique name and it can be seen as an eCAL client.
- **Topic:** The most basic description of the data to be published and subscribed.
- **Publisher:** A Publisher is the object responsible for the actual dissemination of publications.
- **Subscriber:** A Subscriber is the object responsible for the actual reception of the data resulting from its subscriptions.
- **Message:** The message is data itself which is shared in the network.
- **Callback:** A Callback can be used to react on time events, on incoming messages, on service requests or service responses.

Moreover, eCAL has many other extra tools which help the users on the Continental's environment such as a Monitor to see all the data at real-time passing through the network, a Recorder to record any data desired and a Player to replay recorded data and simulate the vehicle's environment at the office. The ROS system has a functionality a bit similar, which allows store data and replay, called "Bags".

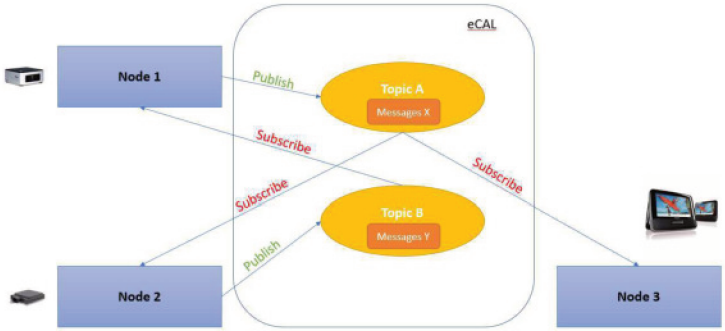


Figure 3.9: An eCAL example

3.6 Hardware Configuration

As an integration work, there were several devices configurations to be done.

3.6.1 Media Gateway

The Technica Media Gateway used on the project provides many degrees of freedom to set the ways where data may pass or not and you can either tag untagged data and untag tagged data, so it can be used as a powerful switch as it is used like a $BroadR-Reach \rightarrow Ethernet$ converter.

Typically, the first Ethernet port is the configuration port and it is also the output for the Switch 1. To enter in configuration mode, you have to set an IP address on the computer that matches the gateway's IP and connect an Ethernet cable between the first port and the computer.

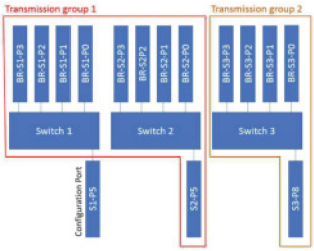


Figure 3.10: Media Gateway's configuration

Each switch can have up to four BroadR-Reach devices connected to it. So the decision was to have three radars connected to the *Switch 1* and three on the *Switch 2*.

On the figure 3.10, we can see the gateway's current configuration. The way it was configured leaves the first port (S1-P5) fully dedicated to configuration, uses the second port (S2-P5) as the output for all the radars data and leaves the third port (S3-P8) available to cast the data from the *Swicth 3*. The idea is to connect four satellites cameras on it in a near future.

3.6.2 PCAN Router PRO

To make an output message matching the expected input, information about how data are encoded on both sides are required. In this case, we used **D**ata **B**ase **C**AN (DBC) proprietary format, which is created by the manufacturer. Its messages characteristics are :

- Byte order (big-endian or little-endian);
- Message length (number of bits);
- Minimum physical representable value;

- Maximum physical representable value;
- Scaling factor (**F**);
- Offset (**O**);
- Unit of measurement (**U**);

Each message in the CAN bus data has a single ID hexadecimal number and a single name. Inside each message, it's possible to have several signals with variable lengths, being each signal a specific information of the vehicle in raw value.

To use these data, it is necessary to transform raw data into physical data on a certain unity of measurement. The DBC contains the information about how to do such a conversion as well.

The physical value can be recovered from the raw value following the equation:

(raw value * F + O) * U

Given an input value V_i, an output value V_o and the unit conversion ratio $U = \frac{U_o}{U_i}$, we can fin the corrected input conversion factor $C_F \frac{F_o}{F_i}$ and the corrected offset $C_O \frac{O_o - O_i}{F_i}$.

Accordingly, the router can be programmed with either the software provided by Peak, PPCAN-Editor 2, or directly in C. Finally, I've chosen to program in C, as the PPCAN-Editor does a conversion of every signal into an unsigned char, so it would take more conversions to reach the final value I wanted. However in C, we can manipulate the signals freely as we want.

Finally ,the manipulated signals were programmed to be sent with a period of $2ms$ and they were converted as follows:

- Vehicle Speed in $[km/h]$
- Transversal Acceleration in $[g]$
- Longitudinal Acceleration in $[m/s^2]$
- Yaw rate in $[°/s]$

3.7 Software Configuration

3.7.1 Timing Synchronization (PTP)

To be able to merge all the radars signals, the Radar Belt system has to set a clock reference through the chain, so the processing unit may know which samples belong to the same moment in time, what it would mean to have the same timestamp.

The chosen clock synchronization for the system was the **P**recision **T**ime **P**rotocol. The computer was set as the only master and all the devices are slaves, hence the computer's clock becomes the clock of every single device connected to the system. The configuration was done by an open source software called *PTPdaemon*.

The synchronization process is divided into two phases. The first one aims to obtain a clock offset correction and then, the second one determines the delay measurement.

The process starts with the "master", which sends a "sync" message via multicast messaging. The "slave" calculates the time difference between its internal cycle and that of the master. In addition, Delay Request Message (by the slave) and Delay Response Message (by the master) are sent to synchronize the slave clock with the master cycle. This process is illustrated on the figure 3.11.

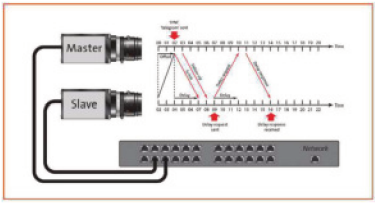


Figure 3.11: PTP synchronization mechanism and delay calculation

3.7.2 Publish VDY data through eCAL

Python is known by its facilities in many scenarios, for instance handle with CAN data in C++ is much more complicated then in Python. For this reason, a Python script was developed to grab the CAN messages, arriving by the USB port, with the VDY data available. The used libraries were *python-can* to handle CAN data and *cantools* for automatic parsing DBC file. Afterwards, the script decodes the messages with a given DBC file, verifies if the messages ID matches with those desired signals, converts the unities if necessary and publishes them through eCAL.

Figure 3.12: Workflow of the script charged to send VDY data through eCAL

Three signals had to have their physical unities of measurement converted, as the Radar Belt system demanded a different unity :

- Vehicle Speed : $[m/s] \rightarrow [km/h]$
- Transversal Acceleration : $[g] \rightarrow [m/s^2]$
- Yaw rate : $[°/s] \rightarrow [rad/s]$

3.8 Misalignment Fix

To better explain the problem and the solution found due to a misalignment between the measured position of the radars and the reality, is necessary to firstly explain how the algorithms determine that a detected object is static.

The radial velocity present on the figure 2.3 is calculated using the vehicle's speed and the yaw angle for static objects. The curve resulted from this calculation must fits on a cosine due to the radial velocity different components. Regarding the six radars around the vehicle, the detected objects must be in different parts of this cosine for each radar, since they all have different yaw angles. This behavior is illustrated on the figure 3.13.

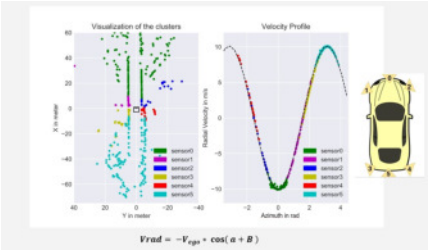
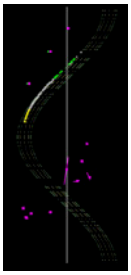


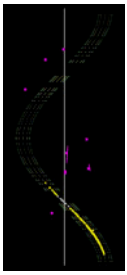
Figure 3.13: Static objects detected and the radial

Furthermore, after all the integration, a new tool, developed by the same team that developed the Radar Belt's additional algorithms, was used to increase the precision of the yaw angle entered as a parameter to these algorithms. On the tool's visualization, the yellow clusters represents the static detected objects, hence they should fit into the cosine curve. If there is a misalignment between mounting parameter enter and the real yaw angle, some static clusters rest outside of the cosine curve, as a consequence, the algorithm considers them as dynamic objects, creating ghost objects on the Centralized Object Tracking, since it only displays dynamic objects.

The visualization of the debug tool is shown in the figure 3.14.



(a) Cosine curve with a bad radar mounting parameter parsed



(b) Corrected image after calibration

Figure 3.14: Cosine curve with a good radar mounting parameter parsed

3.9 Results

To begin this session, here is in introduction of the Radar Belt system visualization tool, which allows us to analyze the recordings, debug the problems and notably see the results of the algorithms.

The visualization framework is called Monitoring Tool and have a single tab for the following visualization modes:

- **Raw detections of a single radar:** The raw detections for a single radar are shown on the figure 3.15. The red circles are detections from the FRS and the black ones from the NRS. We can change the visualization to any specific radar.

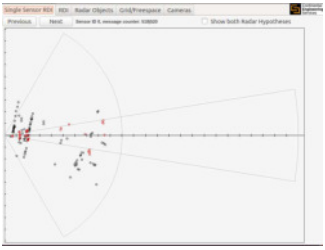


Figure 3.15: Detected objects raw data for a single radar

- **Raw detections for all the radars:** The figure 3.16 shows raw detections as well, but this time the raw detections of all radars are shown on a single image. This is possible due to an initialization file that we give to the algorithm with the mounting parameters (position and yaw angle) of every single radar, then the algorithm places the radars as circles on a map, with arrows indicating their directions and finally, the detections of each radar are shown on the cone in front of them.

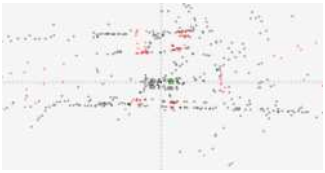


Figure 3.16: Detected objects raw data for all radars together

- **Centralized Object Tracking:** The Centralized Object Tracking shows a box for every dynamic object it detects after post-processing of the raw detections data. The map is following the AUTOSAR coordinates already shown on figure 3.1.

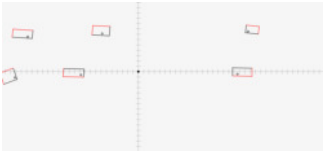


Figure 3.17: Centralized Object Tracking in AUTOSAR coordinates

- **Occupancy Grid and Free Space** This last tab shows the last two algorithms together. The Occupancy Grid creates the green map around the vehicle, where the car is represented by the arrow pointing to its direction. The red points are static objects detected by the algorithm post-processing. Hence, the Free Space use this information to determine where the car may move, that is where there are not red points. On the figure 3.18, we can see an example of the vehicle joining a roundabout.

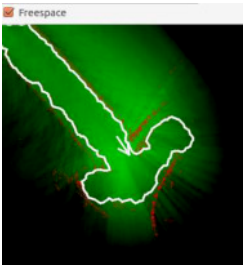


Figure 3.18: Occupancy Grid and Free Space

Now, I am going to give a special attention to the additional algorithms, which represent examples of the powerful tools that are possible to be developed with the Radar Belt system. Both algorithms need VDY data and mounting parameters as mandatory inputs, which are recovered by eCAL and then utilized on the code.

3.9.1 Centralized Object Tracking

The 360° tracking module processes the joint list of radar detections of all radars to estimate the dynamic properties of objects around the vehicle. Compared to fusion concepts where the data of radars are merged on object list level, this aims to improve performance especially in overlap areas of the radars and if objects are detected by different radars over time.

When the algorithm output is displayed on the visualization tool, all currently detected objects are plotted in an AUTOSAR coordinate system. Black lines form the detected L-Shape of the object and red lines illustrate the corresponding rectangle.

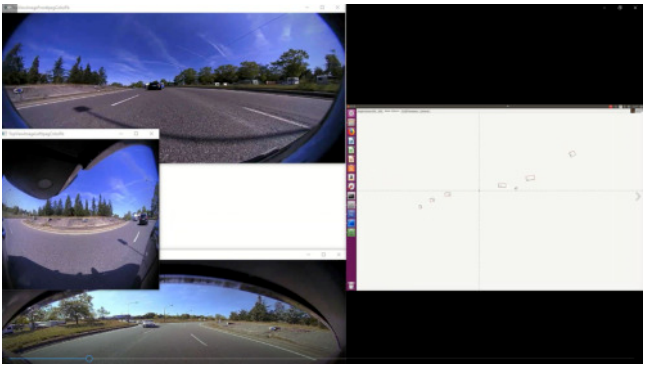


Figure 3.19: Centralized Object Tracking detecting several vehicles nearby

After several times on the road to record the algorithm performance, replay it on the office and try to fix the bugs, we could reach a satisfying result on detecting dynamic objects. The algorithm describes well the size of the objects, being possible to differ cars, bikes and pedestrians. The system does not have blind spots and can detect an object coming from a distance around $150m$.

The biggest difficulty encountered was the match between the parsed mounting parameters and the real ones. The algorithm has shown itself hard to configure, since it demands very precise mounting parameters. The yaw angle is hard to measure on the vehicle with custom tools, therefore the misalignment between measure and reality displayed ghost objects as shown on figure **??**. The cause for this problem was already presented in section 3.8.

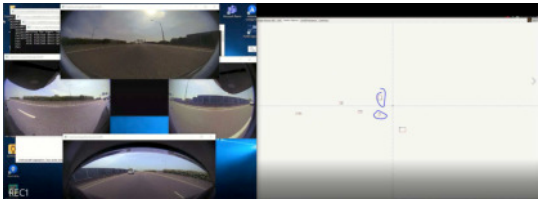


Figure 3.20: Ghost objects appearing on the Centralized Object Tracking algorithm and satellite cameras visualization

The only limitation of the setup are some objects that blinks sometimes, normally they are small. After all the analysis and trying to fix all the errors seen, this problem was the only one we could not solve.

3.9.2 Occupancy Grid and Free Space

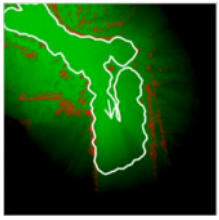
For the representation of the static environment of the vehicle, the radar raw detections are fused in an occupancy grid. First, a radar sensor model is applied to the radar raw detections to map them into a sensor map. By considering some physical effects, the cells between the occupied cells and the sensor are filled with free space values. Finally, the sensor map is fused into the global occupancy grid. The accumulation of all radar detections over time leads to an estimation of the static environment, which is much robust against disturbances like noise, clutter or multipath-effects than single detection image of the radars. The module Occupancy Grid outputs a grid map, in which each cell consists of a probability of occupancy and free space.

In addition, the free space extraction module extracts the free space around the vehicle and represents all the static objects present inside the free space boundary. The output of the free space algorithm depends on the grid, which is input to the algorithm. Free Space algorithm processes the grid map to extract the free space boundary around the vehicle and the static objects inside this boundary. The output of the algorithm is the free space boundary around the ego vehicle.

After several tours on the route to take some recordings, the algorithm performance was really good. On simple roads, the boundaries are well defined; on the routes, it can be seen the free space besides the vehicle, which represents the other roads: on roundabout, all the exits for other roads are also well defined.

However, the algorithms encountered some difficulties in specific cases, like: parking environments, forks and routes with a flowerbed between the two directions. On the fig 3.21, an example of these algorithms is shown.

(a) Satellite camera images used to analyze the recordings



(b) Occupancy Grid and Free Space algorithms visualization

Figure 3.21: Occupancy Grid and Free Space extraction with satellite cameras images

On parking environments, the Free Space could not retrieve the current free space around the car many times. On some forks, the Free Space could not realize there was a road fence between the two recently separated roads. The same problem was encountered with flowerbeds

on the middle of two roads, the Free Space sometimes did not detect the flowerbed and its output represented all the space of the two roads as a free space.

3.10 Conclusion

As a product for clients develop their own algorithms, the main goal of the Radar Belt system was to present a tool to simplify the communication with the radars and provide some libraries to use the radar data and also send VDY and mounting parameters data. Regarding these tasks, it does its work perfectly.

The additional algorithms are created to be commercialized as well, but their main goal is to be used like a showcase to the clients, so they can have a more concrete idea of the powerful tools they can develop with such a system. Therefore, they are not ready, as they have some limitations to be fixed on the code implementation, however they also do their tasks well, because they work in many different scenarios and represent a big step towards the autonomous driving.

Concerning the integration work that was my job, I can say it was a satisfied work that was done, since all the configuration necessary was done, the Radar Belt system works smoothly on the demonstration car nowadays and it is ready to be embedded definitely in the car with a single script that starts, once the vehicle is turned on.

To talk about the future of this project, I would like to give some perspective about the next steps on this long road aiming to reach the autonomous driving. As already said, the DemoCar project is only on the first step, which is acquiring the environment information around the car with different sensors, giving different perspectives and working in different scenarios. Now, as the integration of the radars is done, we can imagine the data fusion between radars and others sensors to reach higher level functions.

Finally, the planned next step is to merge satellites cameras data with radar data. The Technica Media Gateway already used for this project can be used for the cameras as well, as it has a free switch and actually it was already configured to be able to receive data from both sensors. With this fusion, we can use radars to detect objects, as they can detect longer distances and choose a region of interest for the cameras to process the recognition of these objects.

A table with the advantages and disadvantages of each sensor is shown on figure 3.22.

COMPUTER PERCEPTION ANALYSIS

	Camera	RADAR	LiDAR
Object Detection	Medium	High	High
Classification	High	Medium	Low
Density of Raw Data	High	Medium	High
Velocity Measurement	Low	High	Low
Lane Detection	High	Low	Low
Sign Recognition	High	Low	Low
Rain, Fog, Snow Vision	Low	High	Low
Night Vision	Low	High	High
Cost	Medium	Low	High

Figure 3.22: Performance comparison in different scenarios for cameras, radars and lidars

4 Mirror View Project

4.1 Motivation

The environment perception consists on the acquisition of all the data around the vehicle through the sensors utilization. This process is the first part of all the ADAS systems and it is strictly necessary.

The Mirror View is a project of environment perception that allows us to eliminate blind spots and to highlight detected objects over the images. I've developed an application whose the base layer was the internal software : the CES Qt Framework.

4.2 CES Qt Framework

The CES Qt Framework is an internal software developed in C++ and powered by Qt. Its central idea is to help all the developers in CES from several different teams to create their algorithms and functions. It works like several DLLs providing functions commonly used when working with vehicle's sensors.

The Framework brings facilities to play in real-time the algorithms, record the resulted data and then replay the recordings in the office. This is very important as it allows offline modifications, so the developers do not need to have the real sensors at their offices, nor they need to have the car available to test their algorithms.

Other big advantage provided by the CES Qt Framework is its workspace. It is clean and very intuitive to use, in addition it has all the HMI part also available to the developers, so they might profit from a highly developed environment to make and present their demonstrations.

The Framework still has integrated on it generic interfaces to communicate with several CES sensors (GPS, ASL cameras, webcams, etc.) and also the most used communication protocols which are eCAL and CAN.

Aiming to be general to attend several requisites; and robust to have strictly independent

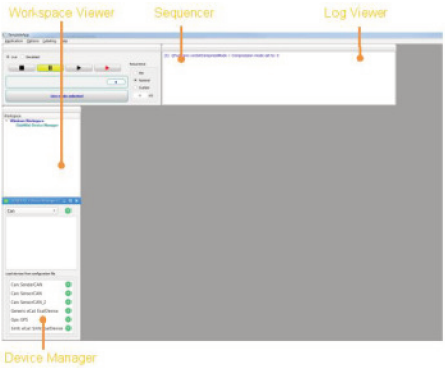


Figure 4.1: Framework workspace

modules, the framework was developed following the **M**odel-**V**iew-**C**ontroller (**MVC**), where the display objects are separated from the business objects and a controller object handles input calls from applications. This architecture allows big modularity, consequently it can work on embedded systems without the display functions, saving power and complexity of the algorithms.

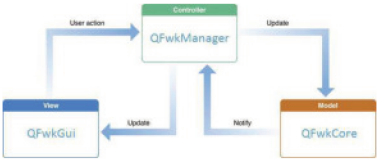


Figure 4.2: Framework's MVC architecture

4.3 Overview

This software aims to simulate physical car mirrors on digital displays using three cameras and monitors as well as optical distortions presented on real mirrors. The intended advantages are to reduce the blind spot and widen the field of view.

To do so, camera video streams are linked to the application which also relies on OpenCV library. Each frame is going to pass though a calibration process, cropped at a Region of Interest size and then, a distortion model is applied.

The three cameras will be positioned at the same place the rear view mirrors are nowadays for coherence purposed. The figure 4.3 shows them :

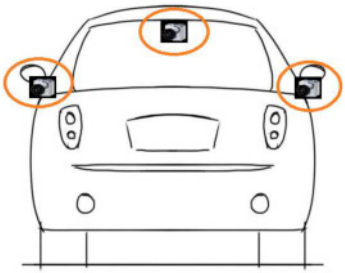


Figure 4.3: Mirror View cameras place on the vehicle

4.4 Calibration

4.4.1 Camera model

Camera calibration is a mandatory step before exploiting information from an image. it aims to correlate 3D coordinates of any points in reference to its 2D coordinates in the image coordinate system. It allows to get accurate measurement information from the image by correcting optical distortions induced by lens and internal electromagnetic interferences.

To do so, a mathematical model of the camera is necessary. There are several models to describe the behavior of the cameras images, but in our case a simple pinhole model is good enough.

- $\vec{X_1}$,$\vec{X_2}$,$\vec{X_3}$ is a coordinate system centered in O (camera pinhole aperture) and $\vec{X_3}$ is oriented in the viewing direction (optical axis).
- $\vec{Y_1}$, $\vec{Y_2}$ is the picture frame coordinate system centered in R which is the projection of O according to $\vec{X_3}$, also called **principal point**.
- The distance f separates the two parallel planes defined by $\vec{X_1}$,$\vec{X_2}$ and $\vec{Y_1}$,$\vec{Y_2}$ and it is known as **focal distance**.
- Assuming a point P with coordinates (x_1,x_2,x_3) in the camera reference, it is projected to Q in the picture frame reference, which is the intersection of (OP) axis and the image plane.

4.4.2 Intrinsic Parameters

Intrinsic parameters describe how light is refracted by the lens system until it hits the imager. These kind of parameters are independent and invariant of moves in space.

The intrinsic parameters encompass focal length, image sensor format and principal point. Using the pinhole model, the projection Q(y_1,y_2) of P(x_1,x_2,x_3) can be written as a combination

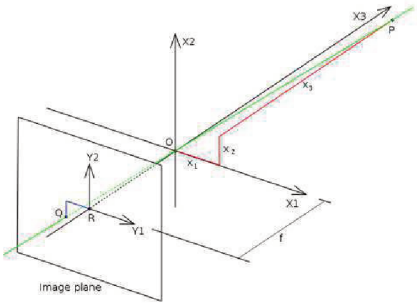


Figure 4.4: Pinhole camera model

of intrinsic parameters and a perspective projection:

$$\begin{bmatrix} y_1 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} k_y 1 * f & s_1 2 & y_1 0 \\ 0 & k_y 2 * f & y_2 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

- k_{y1} and k_{y2} scale factors of the picture in pixels/mm respectively along y_1 and y_2;
- f the focal distance in mm;
- s_{12} a coefficient representing the non-orthogonality row/columns of photoreceptors, being nearly 0 in most of the cases;
- y_{10} and y_{20} the coordinates of the optical center projection in the image plane;

4.4.3 Extrinsic parameters

The extrinsic parameters describe how the optical system is positioned in the space. As this project has not yet been integrated in the vehicle, the tests to develop the application were done at the office, hence extrinsic parameters extraction was not done, since the cameras were not fixed.

4.5 Distortion

To apply the distortion at the boards of the images, a experimental model was purposed.

Parabolic model

The parabolic distortion model come from a constrained second degree polynomial. Its expression is :

$$y = Ax^2 + Bx + C$$

As shown in figure 4.5, distortion is applied to a region of interest that can be set. It starts at s and has a width W. On this area, the minimum magnification factor is mo, at abscissa $s + W/2$.

The distortion shape must fit these three points : (0,1); (W/2,mo); (W,1), considering s as abscissa 0.

Finally, parabola equation coefficients are :

$$A = \frac{4(1 - mo)}{W^2}$$

$$B = \frac{4(mo - 1)}{W}$$

$$C = 1$$

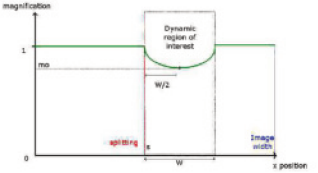


Figure 4.5: Image magnification along x-axis in the region of interest

4.6 Conception

Just like the framework used as the base layer, the application was written in C++. The software used to develop the application was Visual Studio 2015 and the mainly libraries used were notably Qt and OpenCV.

4.6.1 General Workflow of the Application

The tasks of the application were divided like is shown on the figure 4.6. The *QMainTask* is the class where the application starts, accordingly this class manages the framework's most basic tasks to keep working. There is also, in this class, the management of the state signals of the framework, being these signals, Qt signals which trigger slots callback functions whenever they are emitted. The logic behind such implementation is very similar with state machines.

The *ProcessingManager* class is where all the application workflow is managed. All the functions created by the application developer should be called from here.

The distortion part has two main classes. The most important is the *DistortionProcessor*, which it is called in every cycle during the application run-time. It has an initialization function to save the distortion parameters of an instantiated object and its main function is to apply the distortion, but this same function searches for a calibration mapping of the camera to rectifies the image that arrives in the input.

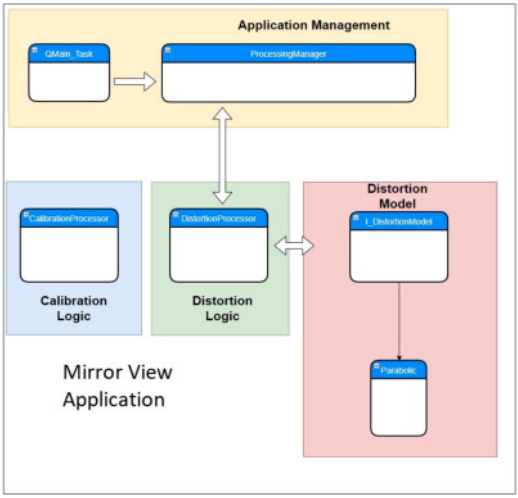


Figure 4.6: Mirror View application scheme

The other important distortion class is the *IDistortionModel*. It is an interface abstract class that was created to allow future developers to test new experimental models of distortion. For now, the only choice available is the *Parabolic Model*. This class is binded to *DistortionProcessor*, so the distortion model can be applied to all the images on the video stream.

The calibration class is apart of the framework functioning. As it is needed an image with well known forms to apply a calibration for a camera model, it is not possible to calibrate a camera at running time. Whenever a new camera will be connected to the application, we should use this class to do the calibration and save the correction mapping files inside the configuration directory.

4.6.2 Configuration file

Aiming to provide configuration availability without touching the internal code, the application has a configuration file .ini written in JSON format. The user can enter with parameters regarding the used cameras, just like their names, their index, enter whether each camera is activated or not and turn on/off full screen mode.

Also, it is demanded to enter the distortion parameters, as position to begin distortion for each side, the four corners of the region of interest and the type of distortion when having several. Lastly, the input arguments for the distortion models. The configuration file can be seen on the figure 4.7.

```
[MIRROR]
Cam_Left=MUI004593
Cam_Right=MUI002355
LeftScreenIndex=1
RightScreenIndex=0
LeftActive=1
RightActive=1
FullScreen=0

[DISTORTION]
l_x_split=600
r_x_split=600
l_x_top_roi=230
l_y_top_roi=80
l_x_bot_roi=1250
l_y_bot_roi=720
r_x_top_roi=30
r_y_top_roi=80
r_x_bot_roi=1050
r_y_bot_roi=720
type=0

[PARABOLIC_DISTORTION]
mag_factor=0.66
```

Figure 4.7: Configuration file of the Mirror View application

4.6.3 Initialization

On the start up of the application, several functions are running on the *init* function of the *QMainTask* and the *Processing Manager*. This functions have the responsibility to initialize all the needed parameters, so the application can work.

All these initialization function are intended to be called only once, if the application runs in a vehicle and it is all in *play mode*. However, they could be called again, if ever the application change to *simulation mode* to read some recordings and then, come back to *play mode*. The figure 4.8 show the workflow of this initialization phase.

Figure 4.8: Workflow of the initialization phase of the Mirror View application

4.6.4 Processing

The workflow of the processing phase from images capture to display are shown on the figure 4.9. The application starts by searching for a correction map of the used camera to correct the incoming images, therefore it is advisable to do the calibration separately before. There is no need to process it more than once, because whenever the intrinsic and extrinsic parameters are found, a calibration mapping to every pixel can be stored in memory. Finally, we only have to access it, instead of calculate the calibration on every single cycle.

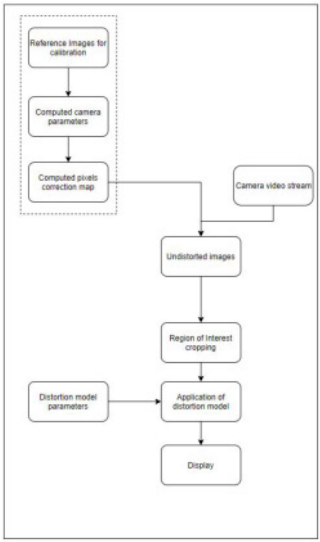


Figure 4.9: Mirror View application workflow

4.7 Real-time constraints

As a real-time application, the processing period for each frame should not pass $40ms$, which is equals to $25 frames per second$. At the middle of the project, with the distortion already being applied, the total time was $150ms$.

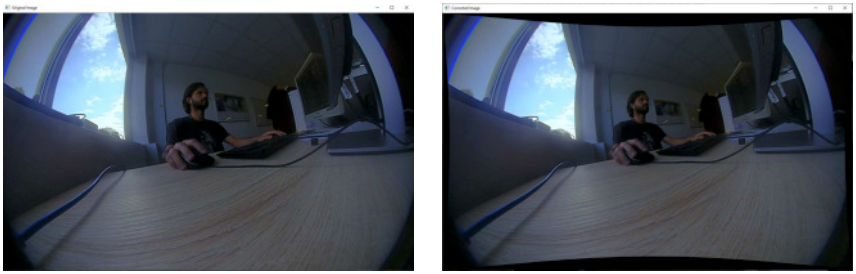
Therefore, some attention was given to the performance and , after some optimizations, the goal was reached in release mode. The function destined to apply the distortion model consumed a lot of time, so the solution found was to share the pointers of some matrices used along the function.

Another optimization that saved us much processing time was on the acquisition of the camera image. Since the framework was doing a conversion from BGR to RGB pixel by pixel, I replaced this function with a dedicated one from OpenCV library. At the end, the expected $25 frames per second$ were reached with two cameras.

4.8 Results

For the both cameras used at the office to develop the application, a calibration was done at the beginning of the work. On the figure 4.10, we can see how the images are displayed after the correction. To take off the black points, it was necessary to apply a region of interest and choose

manually the values which could give us the widest image without black regions.



(a) Original image with the intrinsic distortions presented (b) Corrected image after calibration

Figure 4.10: Comparison between an image before and after correction due to calibration

To highlight the distortion applied, a black line was drawn to indicate where the distortion starts to be applied. On the figure 4.11, it is possible to see how the final images are displayed and also to verify the 25 frames per second with two cameras. A full screen mode is integrated into the application as a boolean in the configuration file, therefor when it is "true", that system starts directly in full screen mode, displaying each image on a given screen. This implementation was done aiming the integration in the car already with the embedded screens.

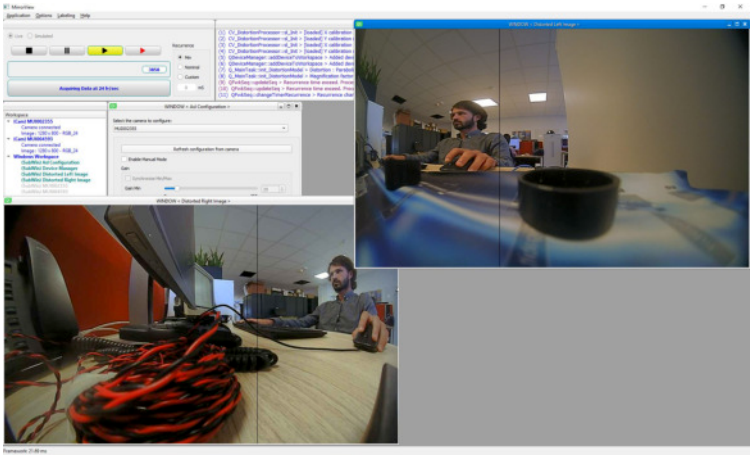


Figure 4.11: Mirror View application final result

4.9 Conclusion

As a bonus project, a lot of work could be done in the Mirror View project. The core of application is done and it runs smoothly in the newest version of the CES Framework with a good video flow with two cameras and a generic input to integrate a third camera.

Still, there is a lot of work to be done. For this reason, the developed code was intended to leave generic structures in case of whether working with one camera, or adding a third one, which would be the rear view. Also, an interface was created to manage the creation of new distortion models.

The distortion models themselves have to be better analyzed, since they did not show a use case where they can bring advantages. The drivers are already familiarized with the distorted borders of the mirrors, but it is not so critically to have only rectified images. However, it is definitively an open point, since there are a thousand of different models that could be already tested.

For a future perspective, it would be very nice to do a HMI to control the calibration of the cameras at running-time. Normally, a client would not change his cameras frequently, however it must not be necessary to change the application at code level like it is nowadays.

Finally, it is exciting the integration of the Mirror View project in the car. It will be a really nice feature that could interact with many other functions, displaying some detected objects or highlighting attention of the driver whenever necessary. There will be a lot of challenges, beginning from how to display three screens, since the embedded computers do not have three display ports. But the fact that it is an application integrated in the Framework could facilitate a lot the job, sending easily the images by eCAL for example.

5 Conclusion

This report has presented in depth the two main stages of this internship. Even having the one main subject for the internship and the other one as a bonus, it was really good to have both projects available for the same internship, because the Radar Belt project, as an integration work, was frequently blocked due to the DemoCar's availability.

Therefore, the tasks were not accomplished on a linear manner as the planning intended, but the interchange between projects allowed me to know a bit of every area the DemoCar team was working on. Talking with some colleagues at CES, I could see that many developers work several years on a project of a sensor that they will never met. Thus, as I am going to keep working with the CES team, it was very valuable to see how the electrical data of the vehicle is managed, working with the CAN bus and retrieving the VDY data, this kind of knowledge will still help me a lot.

In addition, this made me feel more comfortable working on an integration project and testing my electronic skills, learnt through many years of electronic engineering high school, while doing an signal processing specialization. The fact of having the choice to switch to a project of C++ development and image processing gave me confidence to keep on going, testing also some image processing techniques learnt at ENSEEIHT. Finally, for these both projects, I found really awesome to work on the embedded system world, it is very challenging and interesting.

Regarding the perspective for the projects, for the Radar Belt system, there are still a few bugs to fix it, but the work is almost done and it will allow the team to take a step forward and start to merge radar data with cameras. To the Mirror View, the application is ready to start the integration on the car, however there are still many features which could be developed to have a better and more robust final application, also that interacts in run-time with the vehicle's user.

Finally, I appreciate very much to work in an environment as CES, with the Agile method and the tools used there to share information and work together, it was possible to learn and grow a lot as a team member. I am grateful with every colleague in CES, with every teacher in ENSEEIHT and I am really exciting with the next challenges on the steps to merge different sensors data, it seems promising for a fully autonomous driving.