

Sistemas de Tempo Real 2009/2010
Universidade de Aveiro



A Real-Time Framework for the Vision Subsystem in Autonomous Mobile Robots

Fábio Amado 33637

fmna@ua.pt

João Maio 33306

jpsmaio@ua.pt

Índice

1. Introdução
2. A arquitectura de computação CAMBADA
3. O subsistema de visão CAMBADA
4. Arquitectura modular para o processamento de imagem
5. Resultados experimentais
6. Conclusões



I. Introdução

Contexto

- Interesse crescente em agentes móveis autônomos
- Necessária adaptação ao meio
- Tempos de processamento de difícil previsão
- Tarefas : Diferentes níveis de importância



2. A arquitetura de computação CAMBADA

CAMBADA

- Acrónimo de *Cooperative Autonomous Mobile roBots with Advanced Distributed Architecture*
- Robots desenvolvidos na UA que competem na RoboCup
- Têm tido excelentes classificações na competição



CAMBADA

- Coordenar agentes robóticos autónomos para um objectivo comum (futebol)
- Exemplo de interacção com o meio
- Reconhecimento do meio por visão



CAMBADA

- RTAI kernel
- *Low-level sensing/actuating system*
- Rede comunicação de tempo real (CAN)
 - Acesso ao meio determinístico
 - Eficiente uso de largura de banda



CAMBADA

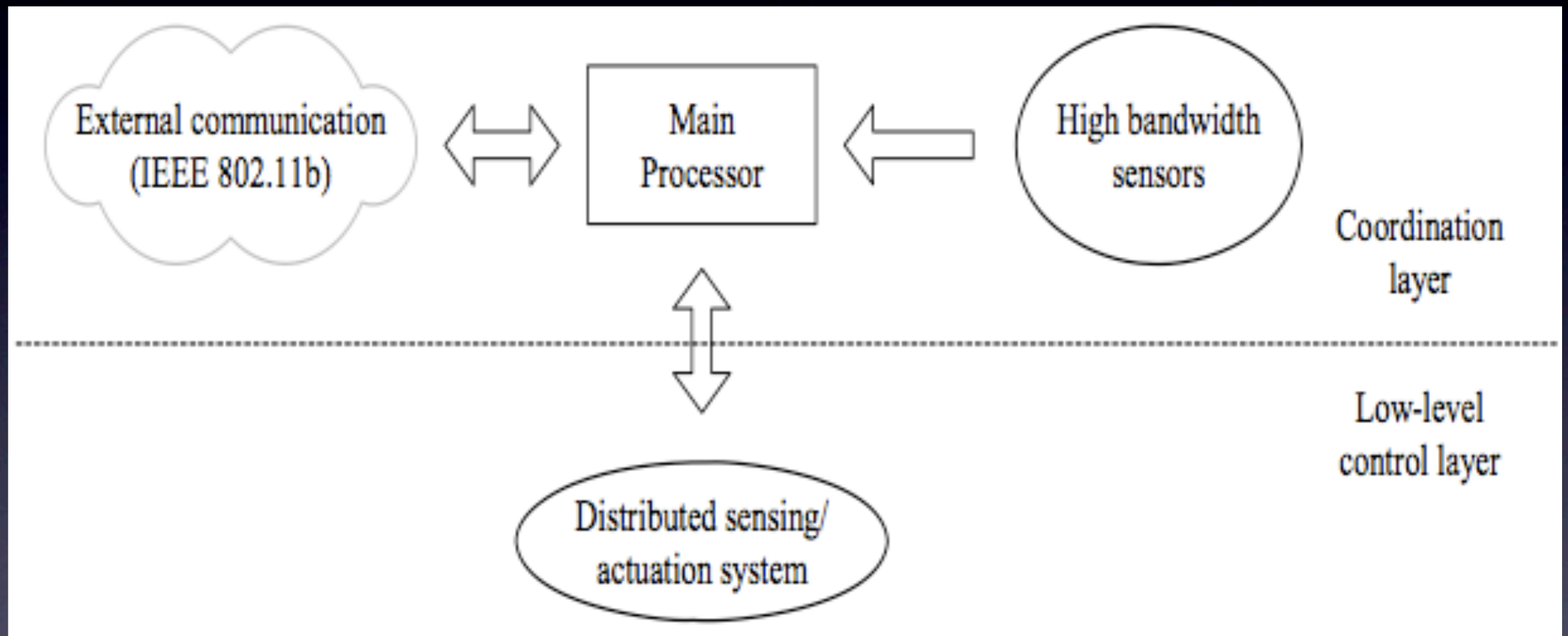


Fig. 1 - Arquitectura dos robots cambada



2. A arquitectura de computação CAMBADA

3. O subsistema de visão CAMBADA

Hardware

- Duas câmaras
 - Direccional - Aponta para a frente (média distância)
 - Omnidireccional - Aponta para o chão (curta distância)
- *Wide-angle lens* (106°)
- 80 cm de altura



Hardware

- 320x240, 640x480 @ 10,15,20FPS
- Interface USB
- Detecção do meio através da cor
- Codificação YUV



YUV

- Espaço de codificação do RGB
- Y - *Luma* (luminosidade)
- U - Diferença no plano azul
- V - Dif. no plano vermelho
- Exemplo para $Y=0.5$
- Sub-espço YMP (*Luma, Module, Phase*)

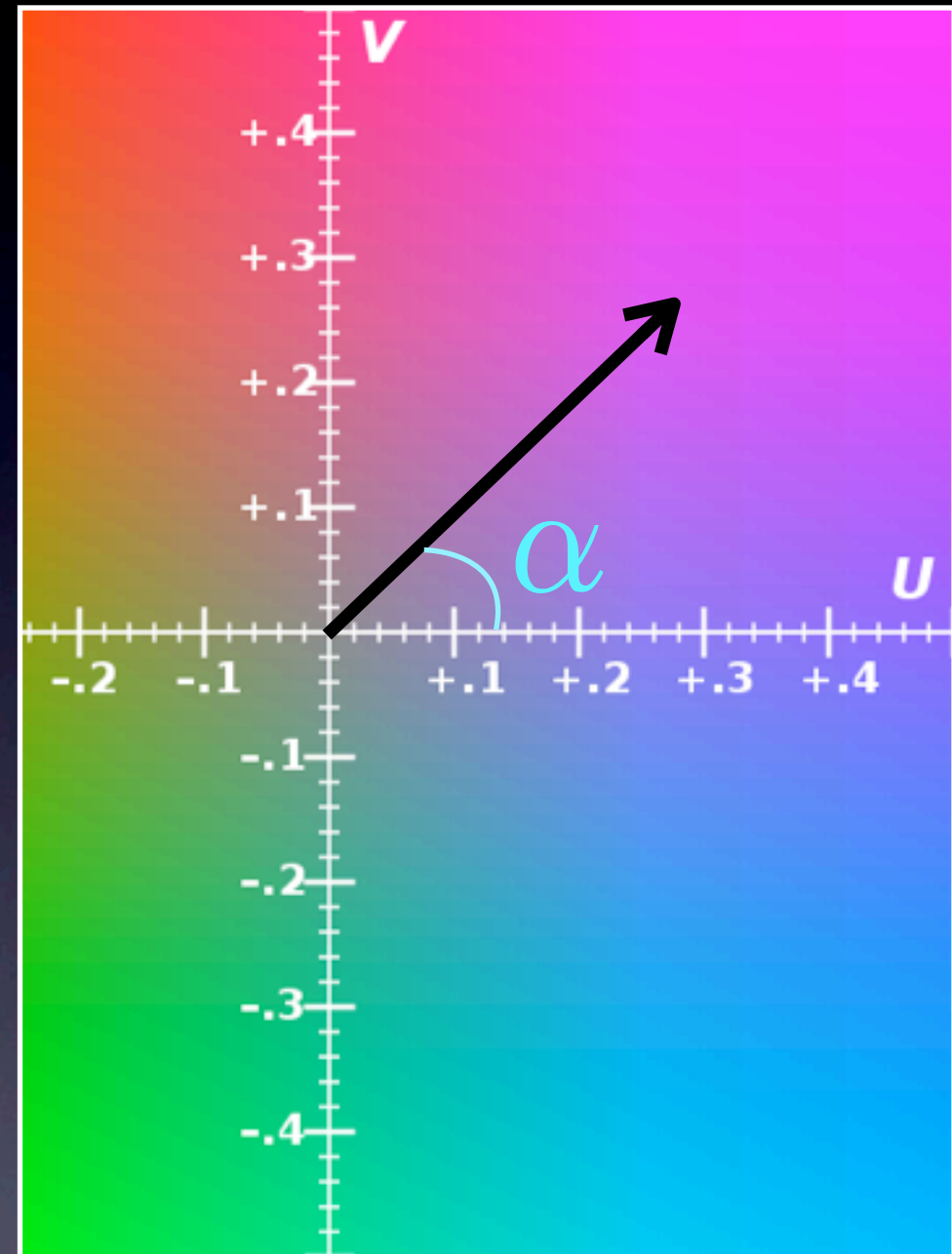


Fig. 2 - Espaço YUV



Processing loop

- Inicialização
- Sleep
- Pesquisa de objectos
- *Update Real Time Database (RTDB)*
- Processar eventos do teclado
- Um processo por câmara

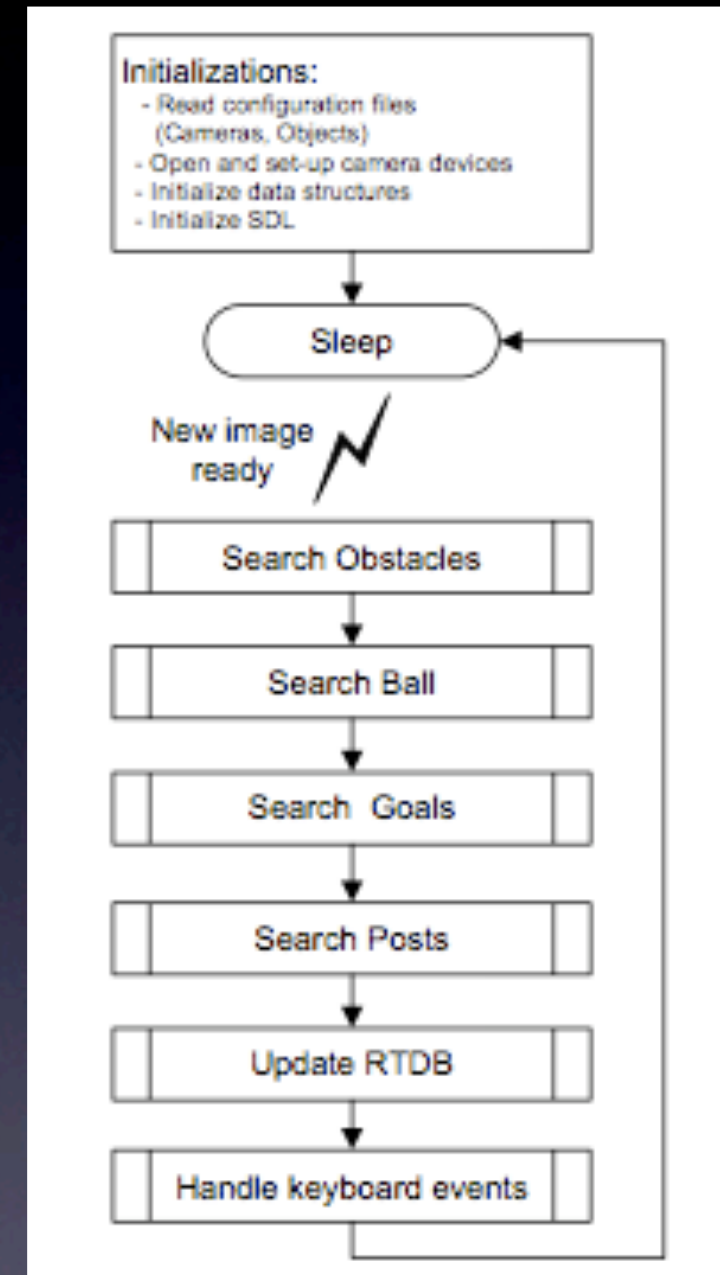


Fig. 3 - Flowchart do processo da câmara direcciona



Front and Omnivision

- *Frame* pesquisado sequencialmente:
 - Obstáculos (outros robots p. ex.)
 - Bola
 - Balizas
 - Linhas de campo
 - Postes demarcadores de campo



Arquitetura monolítica

- Pesquisa dos objectos:
 - Sequencial e independente
 - A partir da última posição conhecida
 - Ou a partir do centro do *frame*



Arquitectura monolítica

- Neste método (actual), em relação ao tempo de processamento:
 - Difícil de prever
 - Esperada elevada variação



Arquitetura monolítica

- 76.1% - menor que 5ms
- 13.9% - entre 25 e 35ms
- Tempo máximo de ~39ms, ~78% do tempo entre *frames*

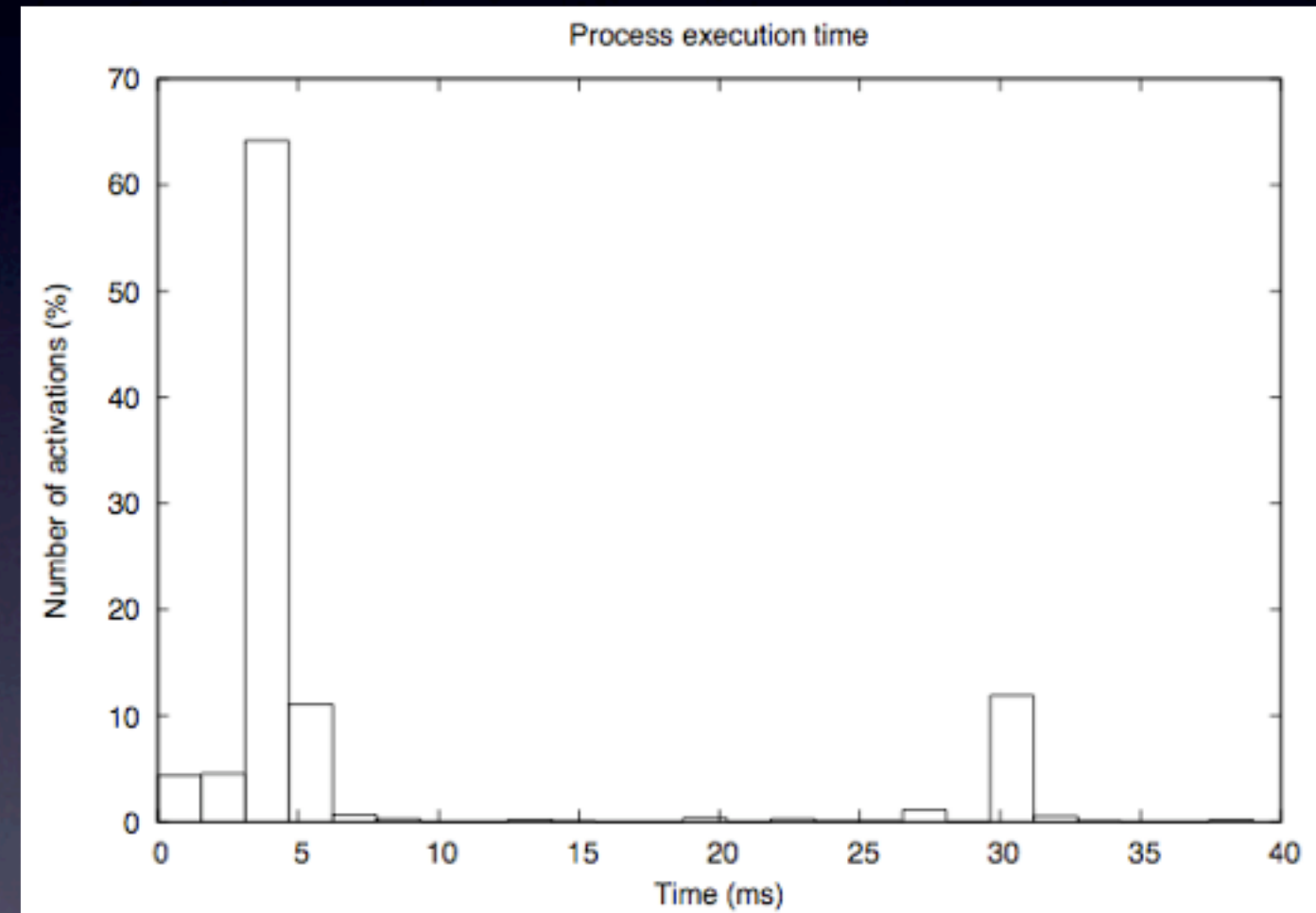


Fig. 4 - Histograma do tempo de pesquisa da bola



Arquitetura monolítica

- Um processo por câmara
- Pesquisa sequencial
- Tempos com elevada variação
- *Worst-case scenario* : centenas de ms
- *Frame skipping*
- *Performance* pobre (blackout)



Solução?

- Aumentar o poder de processamento?
- Co-processadores específicos?
- *Robots* operados por baterias...
- Não é claramente a solução!



4. Uma arquitetura modular para processamento de imagem

Porquê e como?

Técnicas de tempo real

- A latência limita a velocidade máxima do robot (evitar obstáculos)
- Obstáculos mais prioritários que posição
- Métodos de predição + Odometria
- Tarefas menos prioritárias não podem bloquear as mais prioritárias!



Arquitectura modular

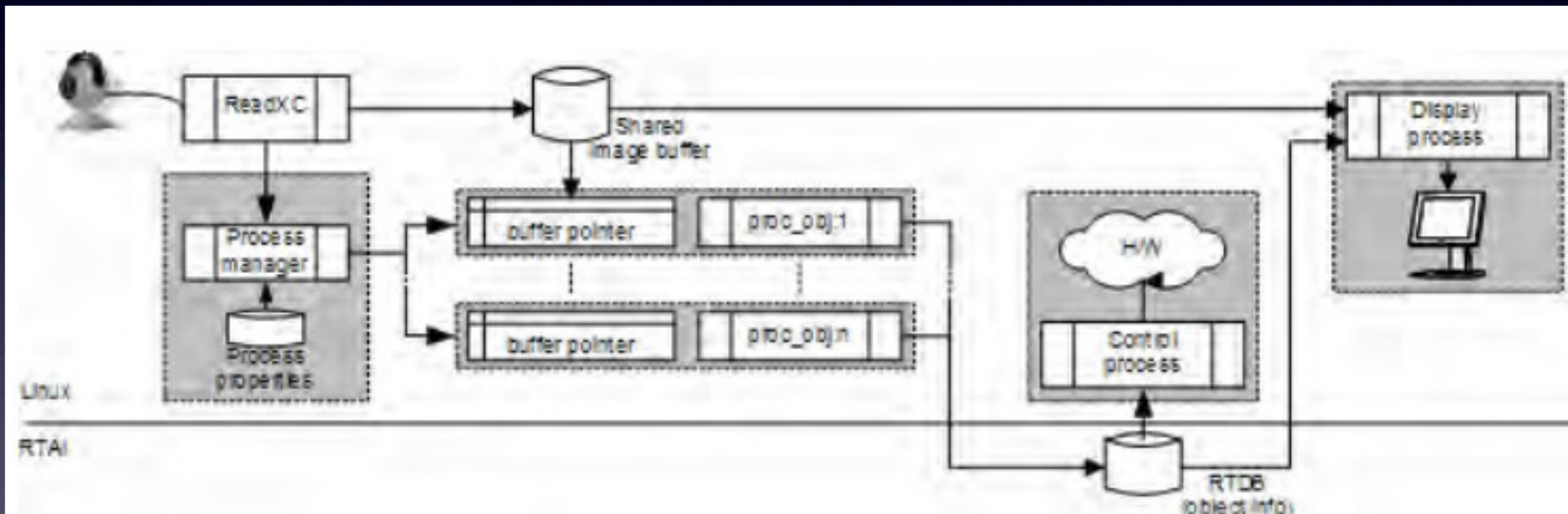


Fig. 5 - Arquitectura modular do sub-sistema de visão CAMBADA



4. Uma arquitetura modular para processamento de imagem

Process Manager

- Tabela com informação relativa aos processos (prioridades, deadlines, estado actual, etc)
- Guarda informação estatística



Shared Data Buffer

- Permite acesso assíncrono e não bloqueante aos *frames*
- Responsável por coerência da informação



QoS Manager

- Meio difícil de prever em *pre-runtime*
- Permite reconfiguração dinâmica



5. Resultados Experimentais

Computador Utilizado

- Intel Pentium 3 550Mhz
- 256MB Ram
- Linux 2.4.27 Kernel
- RTAI 3.0r4
- 2x Logitech Quickcam - 320x240 @ 20FPS



Arquitetura monolítica (I)

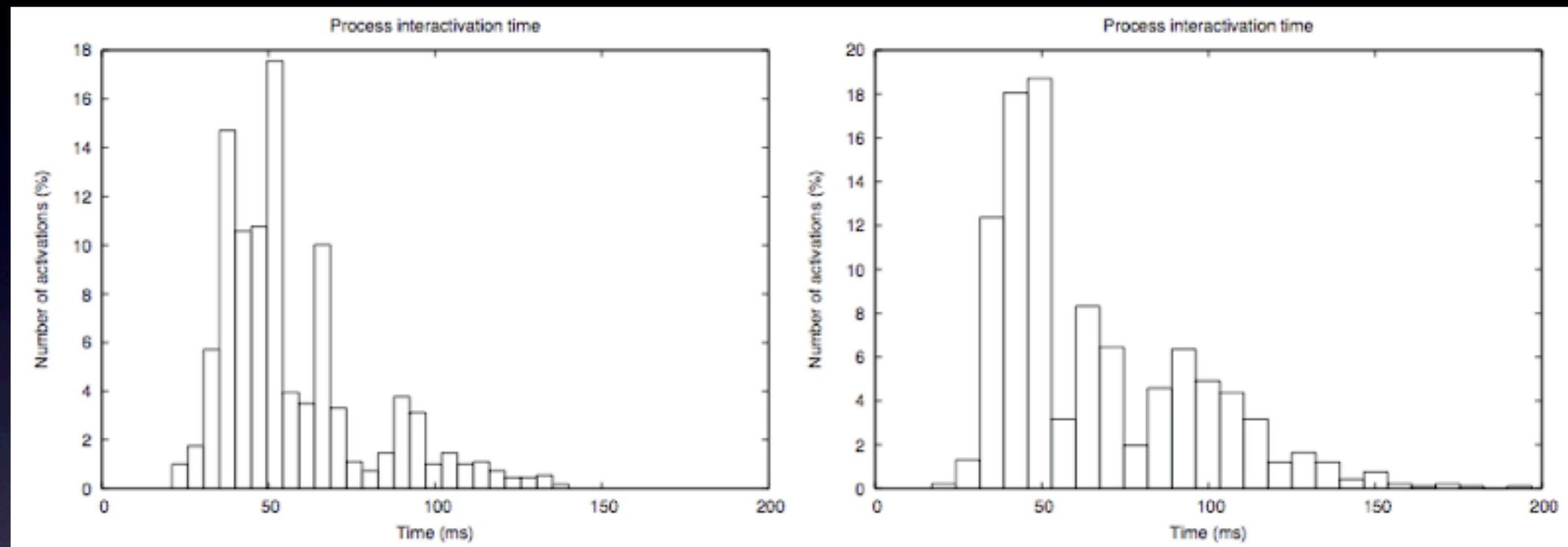


Fig. 6 - Histogramas dos processos FrontVision(esquerda) e OmniVision(direita)

Process	Max. (ms)	Min. (ms)	Avg. (ms)	St.Dev. (ms)
FrontVision	143	29	58	24
OmniVision	197	17	69	31

Fig. 7 - Tabela com tempos de inter-activação dos processos FrontVision e OmniVision



5. Resultados Experimentais

Arquitetura monolítica (II)

- Entre activa  es:
 - *Jitter* elevado
 - Valores entre 17 e 200ms (OmniVision)
 - Valores entre 29 e 143ms (FrontVision)
- 50ms entre *frames*
- *Performance* pobre (blackout)



Arquitetura modular (I)

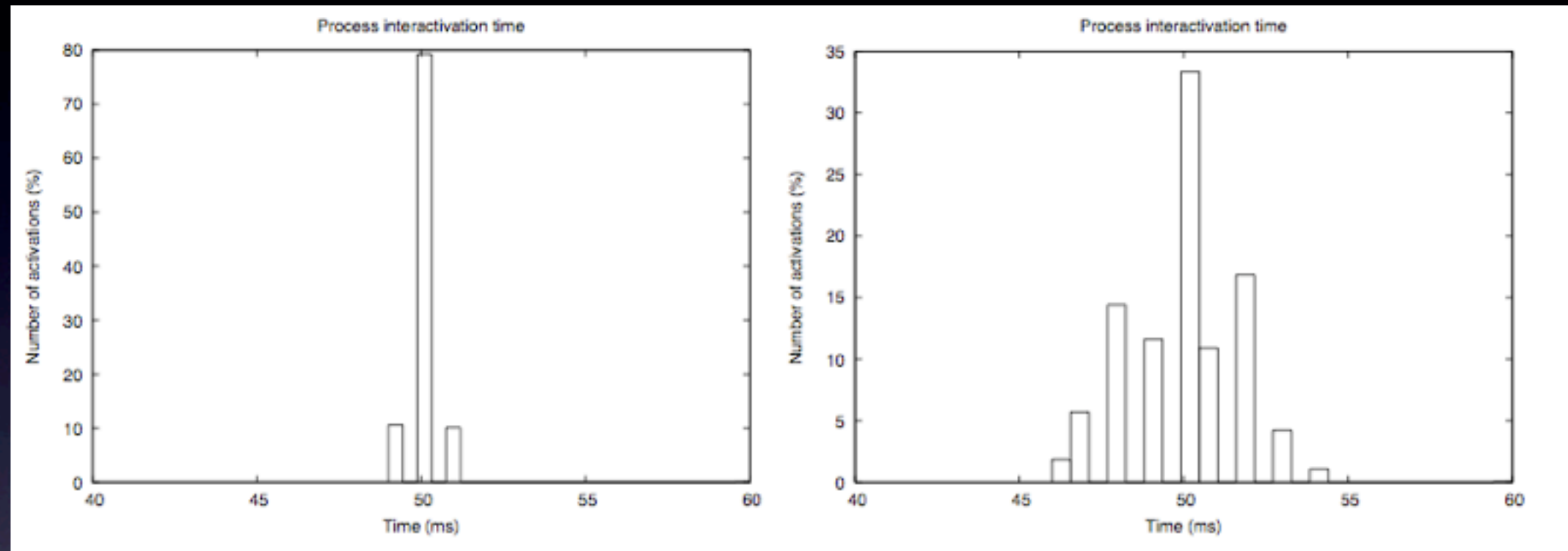


Fig. 8 - Histogramas da detecção de obstáculos na câmara frontal(esquerda) e omnidireccional(direita)

- Pesquisa de obstáculos:
- Tempo de execução menor que 5 ms!
- Evita obstáculos com eficácia



Arquitetura modular (II)

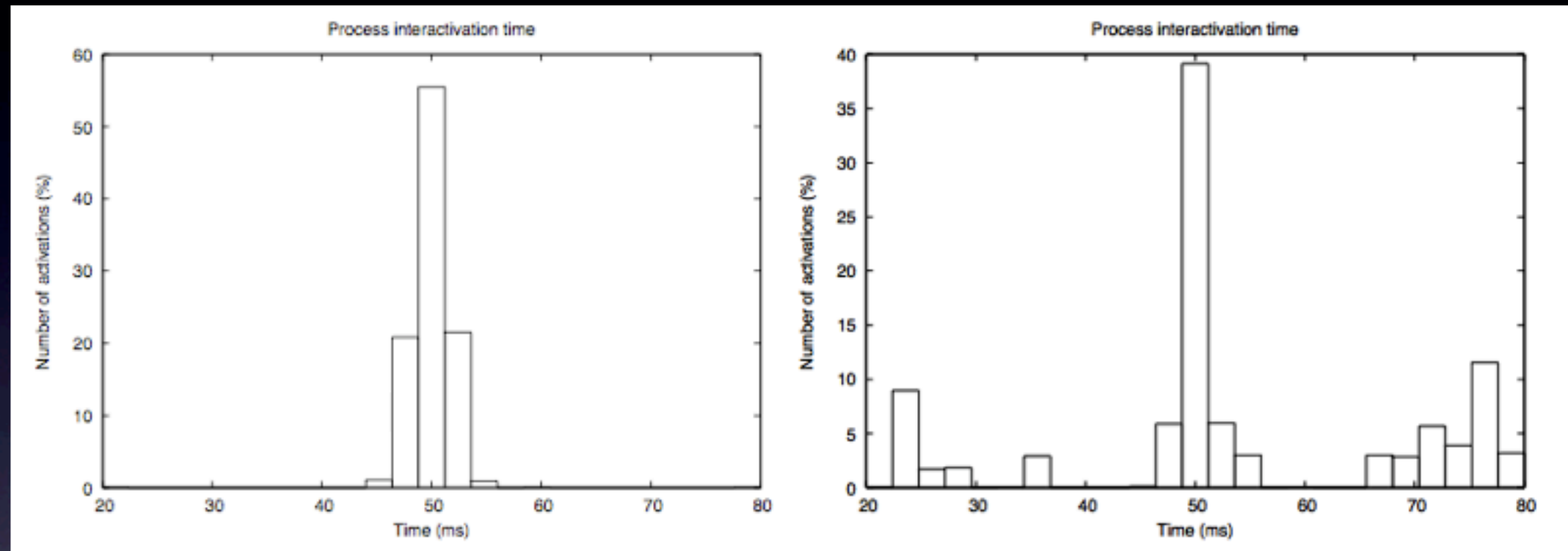


Fig. 9 - Histogramas da detecção da bola na câmara omnidireccional(esquerda) e frontal(direita)

- Pesquisa da bola:
- Tempo de execução menor que 10 ms (omnidireccional)



Arquitetura modular (III)

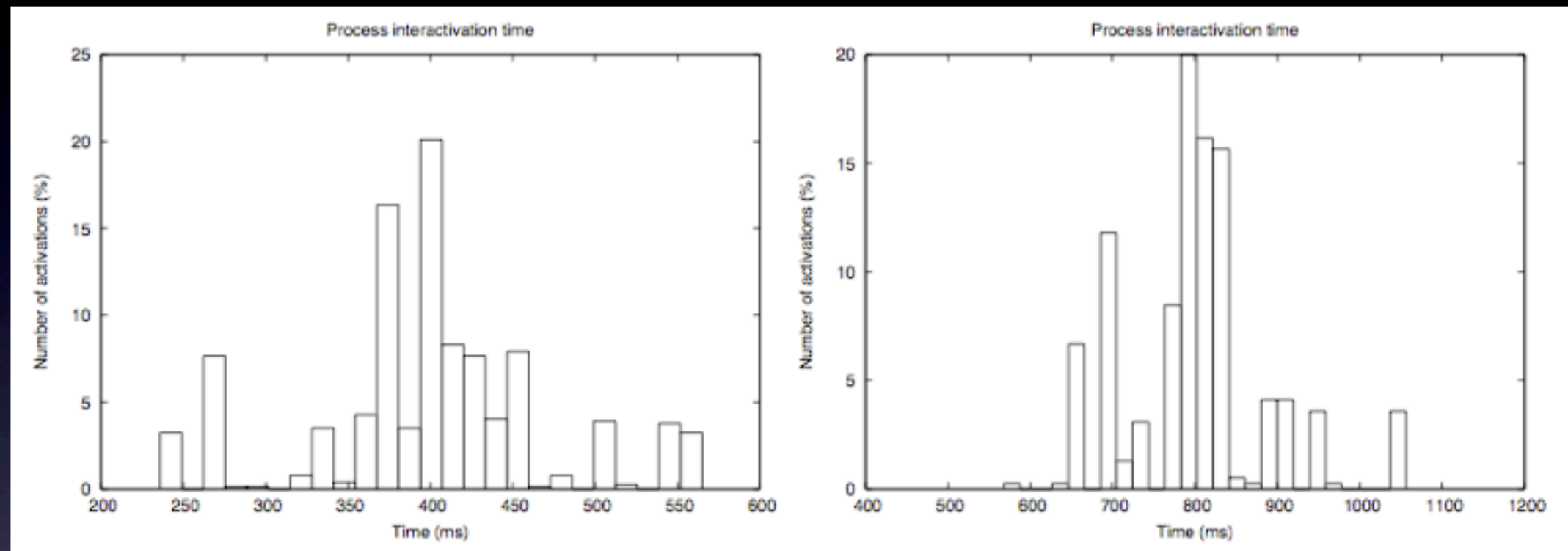


Fig. 10 - Histogramas da detecção das linhas de campo(esquerda) e yellow post(direita)

- Pesquisa de linhas e poste amarelo:
- Não é tão importante



Arquitetura modular (IV)

Process	Max. (ms)	Min. (ms)	Average (ms)	Standard deviation (ms)
Avoid_Fr	60.1	48.9	50.0	0.5
Avoid_Om	60.1	45.9	50.0	1.6
Ball_Om	60.1	46.0	50.0	1.6
Ball_Fr	80.0	19.9	50.0	2.1
Ygoal	362.2	61.1	207.9	58.3
BGoal	383.9	60.9	208.4	66.6
Line	564.7	235.6	399.9	71.9
BPost	1055.8	567.9	799.9	87.2
YPost	1156.4	454.4	799.6	114.3

Fig. 11 - Tabela com tempos de inter-activação e respectivas variâncias de cada processo

- Um *frame* a cada 50ms - Os 4 primeiros processos executam uma vez por *frame*
- Não houve *frame skipping*



Adaptação dinâmica de QoS (I)

- Ganho de posse de bola - Troca de função de um robot
- Alteração “do que é mais importante”
 - Desprezar a procura da bola
 - Melhorar a informação àcerca da própria localização



Adaptação dinâmica de QoS (II)

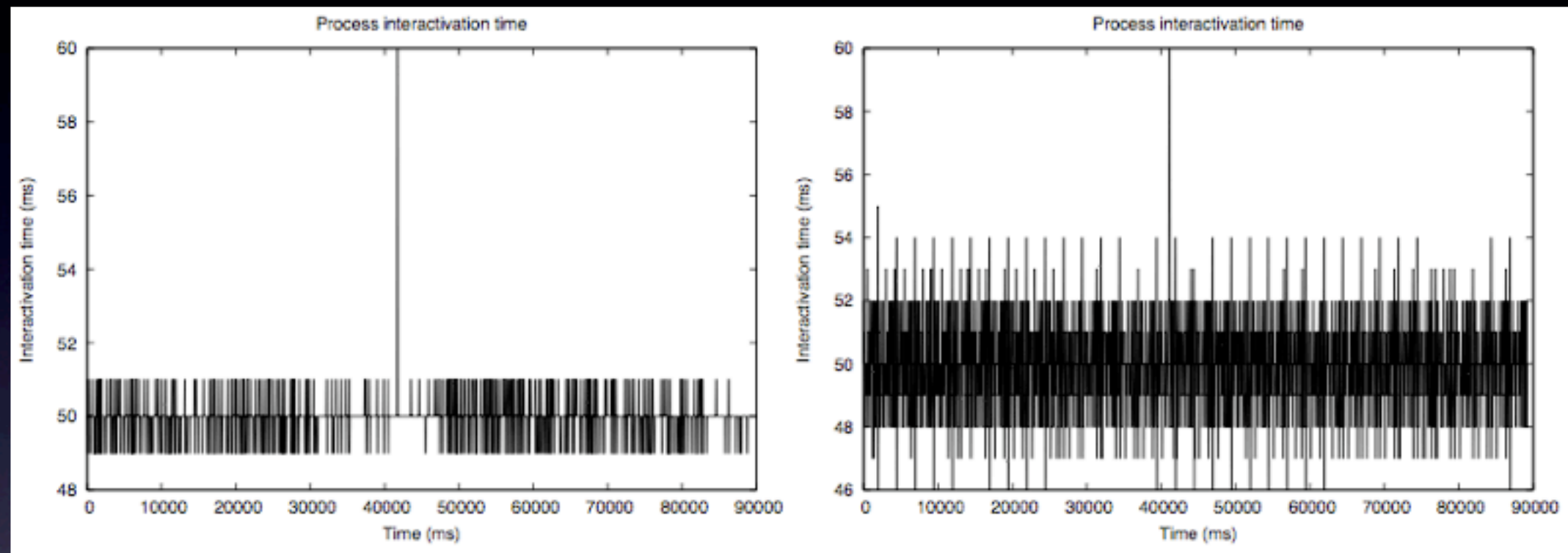


Fig. 12 - Histogramas da detecção de obstáculos na câmara omnidireccional(esquerda) e frontal(direita) com QoS dinâmico

- Picos de activação
- *QoS Update*



Adaptação dinâmica de QoS (III)

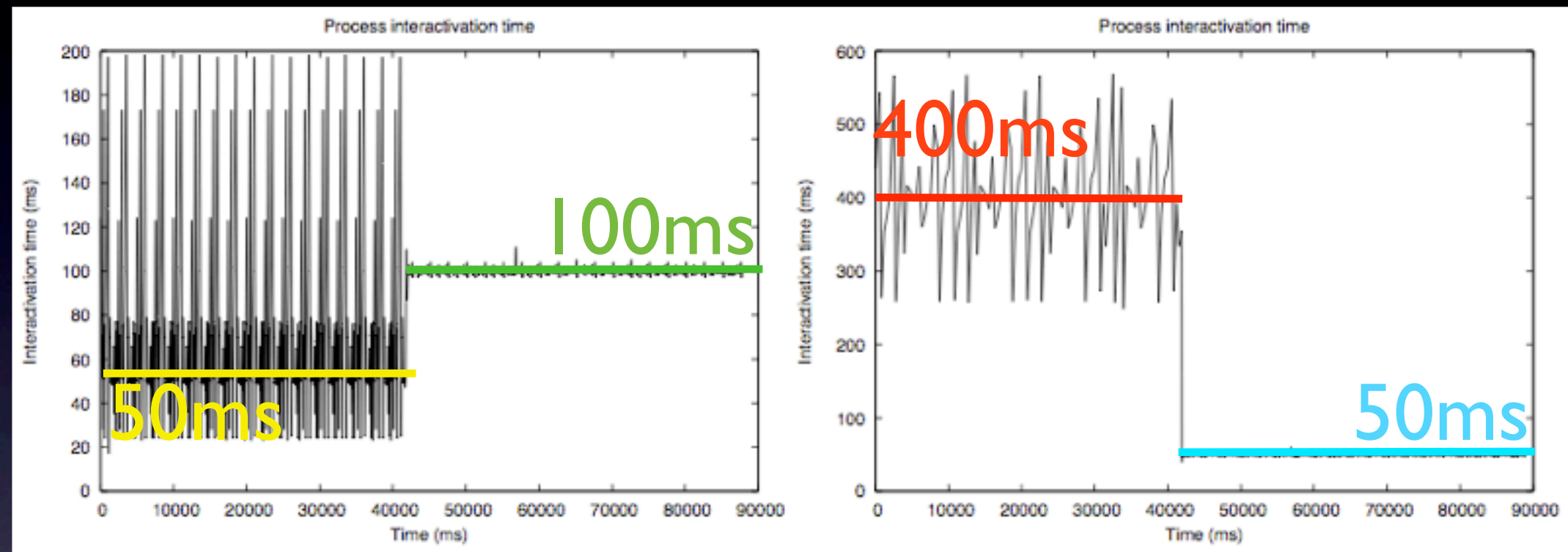


Fig. 13 - Histogramas da detecção da bola(esquerda) e detecção de linha(direita) na câmara frontal com QoS dinâmico

- *QoS Update* aos ~40000ms
- Redução da importância da procura da bola, aumento da procura de linha



6. Conclusões

Conclusões

- Maior parte dos sistemas *vision-based* não se baseiam em técnicas de tempo real
- Pobre performance
- Nova arquitectura modular permite resolver muitos dos problemas
- A arquitectura proposta permite ainda *dynamic QoS*



Referências

- Figura do slide inicial
- Figuras 1,3,4,5,6,7,8,9,10,11,12,13
- A Real-Time Framework for the Vision Subsystem in Autonomous Mobile Robots (Paulo Pedreiras, Filipe Teixeira, Nelson Ferreira, Luís Almeida, Armando Pinho e Frederico Santos UA DETI)
- Figura 2 <http://en.wikipedia.org/wiki/YUV>



Bibliografia

- <http://intechweb.org/downloadpdf.php?id=347&PHPSESSID=kbh9474qqmasbgbvkogee13js6> - *paper* base da apresentação
- <http://en.wikipedia.org/wiki/{robocup,YUV,YCbCr,odometry}>
- www.robocup.org
- <http://www.ieeta.pt/atri/cambada/index.htm>



Questões

