

Aplicação de métodos clássicos de processamento de imagem na detecção e localização de articulações no tronco humano

A partir de uma sequência de imagens/vídeo capturadas com uma câmera RGB-D (cor + profundidade - vídeo e imagens anexados), é pretendido o desenvolvimento de um algoritmo para a detecção de vários pontos de interesse no tronco humano, com particular relevância para os ombros, ancas e tronco (aprox. centro de gravidade), utilizando para isso métodos e ferramentas de processamento de imagem clássico. A detecção em tempo real destes pontos de interesse é de particular relevância na avaliação da estabilidade postural durante o processo de marcha, e pode ser utilizado para seguir a evolução de pacientes com limitações ao nível da marcha e equilíbrio (p.e. casos clínicos de ataxia pós-enfarte).



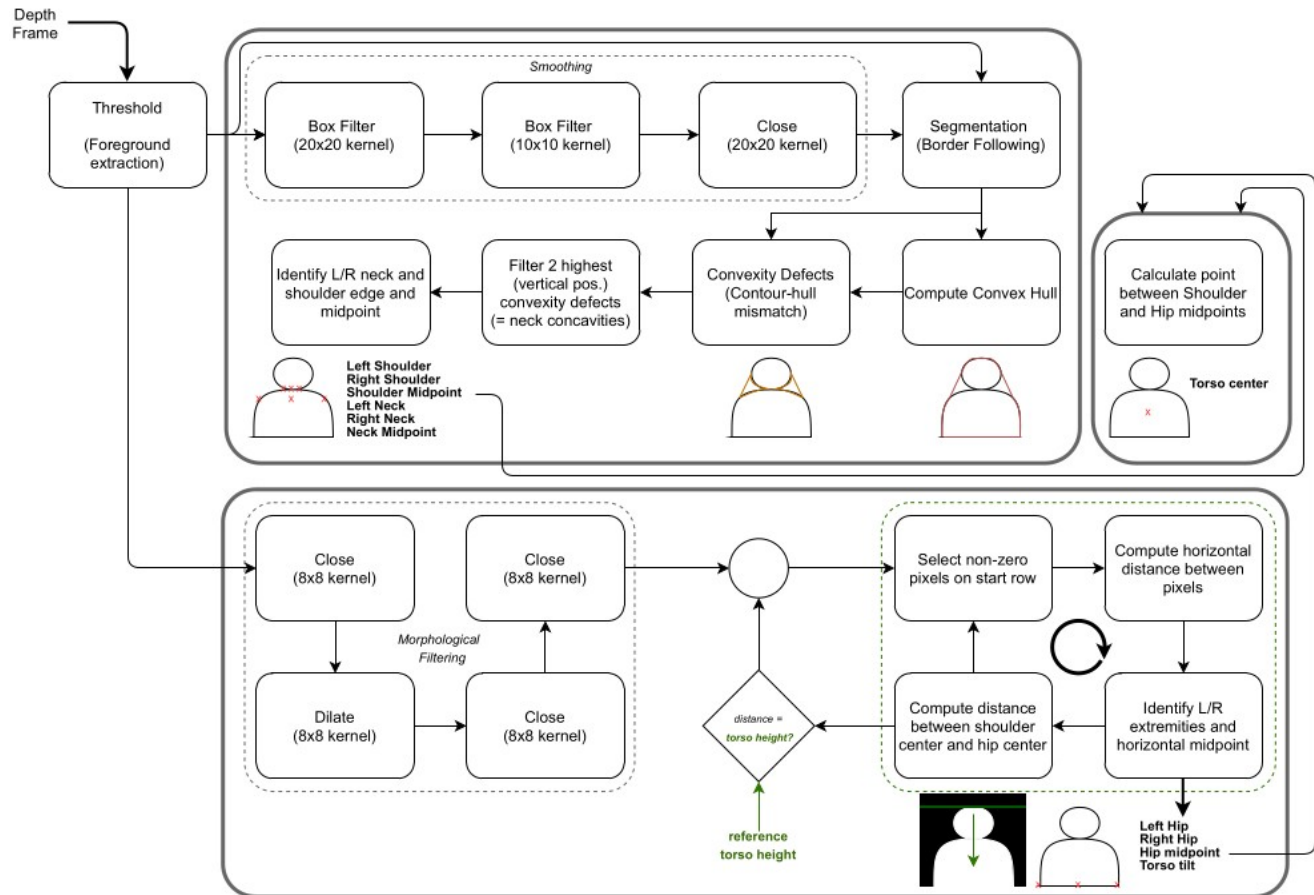
O trabalho incidirá sobre dados adquiridos com sujeitos saudáveis na utilização de um andarilho inteligente equipado com duas câmaras RGBD convencionais (tipo Kinect), que incluem **imagens de cor** (vídeo RGB) e de **profundidade**, pelo que várias abordagens podem ser adotadas de modo a obter os resultados pretendidos.

O resultado pretendido são as posições 2D (linha, coluna) dos píxeis associados com o **pescoço**, **ombros** (direito e esquerdo), **ancas** (direita e esquerda), e **centro do tronco** (Figura 2). Serão fornecidos valores de referência “ground truth” para que os alunos possam validar/quantificar a



distância entre as posições encontradas e o valor real. Opcionalmente, a avaliação pode também ser feita em 3D através da reconstrução das coordenadas mundo com os parâmetros intrínsecos das câmeras, de modo a inferir se a solução se aproxima do valor referência ($\leq 3\text{cm}$ "state of the art").

Exemplo de solução:



É também fornecida uma solução funcional que se espera servir como exemplo/modelo a seguir no delineamento/desenvolvimento de uma nova solução que enderece as limitações atuais. O código foi desenvolvido em C++ para sistemas Unix com recurso às bibliotecas OpenCV que fornecem ferramentas de processamento de imagem.

Todo o processamento de imagem se encontra assente na classe *TorsoDetector*, que implementa através de funções estáticas os vários blocos apresentados na Figura 3. De modo a correr a deteção dos pés deve ser utilizado o método *TorsoDetector::detect()* que retorna um objecto do tipo *std::vector<cv::Point2i>* com os pontos associados aos pontos de interesse que se pretende detetar.

A parametrização do algoritmo é feita através de uma estrutura do tipo *cv::TorsoDetectorParameters*, que permite alterar os vários parâmetros que influenciam o funcionamento/processo de deteção (e.g. dimensão de kernels morfológicos, dimensões referência do sujeito, thresholds para profundidade e área de contorno, etc.).

Conceptualmente, o algoritmo utilizado encontra-se ilustrado na Figura 3; Inicialmente é feita uma operação de *threshold*/remoção do “background”, que permite isolar os pixels relativos ao tronco do sujeito. Seguidamente é aplicado um filtro Gaussiano (*cv::boxFilter*) à imagem binária (*máscara*) resultante do passo anterior, que resulta em um “alargamento” do tronco e acentua as concavidades do pescoço. Depois é extraído o contorno externo convexo (*cv::convexHull*) e identificadas as

concavidades (através do ponto inicial, final e mais distante) correspondentes ao pescoço, a associando, respectivamente: o ponto médio entre os pontos mais distantes do lado esquerdo e direito à articulação do pescoço, e os pontos inicial e final das concavidades aos ombros direito e esquerdo.

A partir da imagem de profundidade inicial, são adicionalmente aplicados filtros morfológicos (*cv::morphologyEx*), e a imagem percorrida de baixo para cima de modo a encontrar a linha que dista do pescoço um valor de referência correspondente à altura do tronco do sujeito.

Paralelamente, é também fornecido código que carrega e sincroniza as imagens de cor e profundidade, e que se espera que seja utilizado caso os alunos decidam desenvolver a solução em C++, ficando a cargo dos mesmos a implementação de mecanismos de sincronização (caso sejam necessários) noutras linguagens de programação. De modo a obter imagens sincronizadas, basta instanciar um objecto da classe *TrialData* fornecendo o caminho para onde os dados se encontram guardados e chamar o método *TrialData::next()* que retorna um *std::vector< cv::Mat >* com as imagens cada sensor.