Interface entre o framework ROS e a plataforma NAO

Cristiana M. Farias, Felipe M. Dalosto, Yuri G. Rocha, Mariana C. Bernardes

Laboratório de Automação e Robótica (LARA)
Universidade de Brasília, Caixa Postal 4386
CEP 70919-970, Brasília-DF, Brazil
E-mail: cristianafar@gmail.com

***Abstract*— Esse trabalho visa realizar a análise de um sistema criado para o robô NAO, produzido pela empresa Aldebaran Robotics, com arquitetura de sistema baseada em ROS (Robot Operating System). Serão também relatados os experimentos efetuados no ambiente da competição RoboCup SPL, assim como seus resultados e expectativas futuras.**

I. Introdução

Dentre os robôs humanoides comercializados atualmente destaca-se o NAO, da Aldebaran Robotics, que com sua aparência cativante e grande gama de sensores e atuadores, o tornam uma ferramenta poderosa para diferentes aplicações. Ele é equipado com um processador Intel Atom de 1.6 GHz e 1 GB de RAM, sistema operacional baseado na distribuição do Linux, Gentoo, além de possuir também 25 graus de liberdade em seus movimentos.

Devido a sua versatilidade, o NAO vem sendo utilizado em diversas áreas de pesquisa pelo mundo, notavelmente interações com crianças autistas [1], robótica educacional [2], interação humano-robô, além, é claro, de participar da categoria de plataforma padrão da Copa do mundo de Robôs, a RoboCup. Este tipo de competição fomenta o desenvolvimento de técnicas de navegação, visão computacional, controle de marcha, tomada de decisões, planejamento de trajetórias, dentre outras áreas de pesquisa em robótica com possibilidade direta de uso em diversas aplicações.

Entretanto, os diferentes tipos de arquitetura usados no desenvolvimento de robôs humanoides dificulta o aproveitamento de códigos entre plataformas e a utilização de certas ferramentas nos diversos sistemas. Com o intuito de simplificar o desenvolvimento e possibilitar maior integração em termos de software, ferramentas tem sido desenvolvidas, como o framework ROS (Robot Operating System) [3].

O ROS é um sistema open source baseado em princípios de comunicação de ponto a ponto, com suporte para diversas linguagens de programação e construção modular de suas bibliotecas. A sua implementação se dá com base em nós, que são processos modulares, ou seja, elementos do sistema realizando alguma função especifica, por exemplo gerenciar o processamento de imagens obtidas de uma câmera. Os nós se comunicam através da publicação e subscrição de mensagens ou através da requisição e resposta de serviços entre eles.

A. Trabalhos relacionados

O NAO é uma plataforma popular e amplamente difundida em diversas instituições de pesquisa, assim como o framework ROS. A combinação de ambos tem sido investigada em trabalhos recentes do Laboratório de Automação e Robótica (LARA) da Universidade de Brasília. Visando o controle de robôs por teleoperação [4], fez-se uso do sistema de serviços e mensagens do ROS para integrar o NAO com um aparelho Kinect, usado para a captação de movimentos. Em outro trabalho [5] que também integrou o ROS, o NAO e o Kinect, foi desenvolvido um método de raciocínio formal combinado com algoritmos de percepção, controle de juntas e tomada de decisão para manipulação de objetos.

Finalmente, a equipe UnBeatables, que trabalha com robótica humanoide no âmbito da RoboCup Standard Plataform League (SPL) investigou o uso do ROS em aplicações de futebol de robôs, a exemplo de outra equipe da SPL [6]. O presente trabalho apresenta os resultados de tal estudo.

B. Standard Platform League (SPL)

A RoboCup SPL é uma iniciativa internacional que fomenta a pesquisa e desenvolvimentos em diferentes áreas da robótica. Para tal, são promovidas partidas de futebol entre robôs completamente autônomos e idênticos, em que é utilizada a plataforma padrão NAO.

As regras do jogo se assemelham as de uma partida de futebol jogada por humanos. Primeiramente temos que o hardware do robô não pode ser alterado: o desenvolvimento deve ser feito somente em nível de software. O campo mede 9 m $\times$ 6 m, tendo as mesmas linhas encontradas em um campo de futebol tradicional. Além disso, a bola adotada é de cor laranja e tem 65 mm de diâmetro e o gol possuí traves amarelas com 80 cm de altura. Também semelhante ao futebol de humanos, a SPL possuí regras referentes a faltas, em que o robô é penalizado ao segurar a bola, derrubar outros jogadores ou ficar inativo por muito tempo.

Na RoboCup se adota um agente de comunicação externo: o Game Controller. Este é controlado por um juiz humano e informa aos robôs os diversos estados do jogo. Assim, os robôs podem se posicionar para o início de uma partida ou retornar ao seu campo após um gol.

C. Objetivos e contribuições deste trabalho

Este trabalho se destina ao cenário de jogos de futebol humanoides da SPL, tendo como objetivo a construção de um

sistema baseado em ROS que atue de maneira satisfatória em uma partida. Será feita a descrição do sistema criado, além da análise de seu funcionamento.

II. Metodologia

A. Configurações do ambiente

O processo de integração do NAO com o ROS foi baseado em métodos de compilação cruzada propostos no trabalho da Universidade do Chile [6] e na documentação do ROS.

Para o setup inicial do sistema são necessários os seguintes requisitos: Ubuntu 12.04, ROS Fuerte, NAO operando com o Naoqi 1.14[1], Naoqi toolchain e Máquina virtual VMWARE. Inicialmente a máquina virtual deverá ser configurada como ambiente nativo para instalação (ao final o sistema instalado será copiado para o robô). Em seguida, deverão ser configuradas as variáveis de ambiente, paths e diretórios a serem utilizados para uma compilação cruzada pelo ROS, além de se incluir a toolchain do NAO dentro da toolchain do ROS. O próximo passo será copiar as bibliotecas *log4cxx.so* e *aprutil*.so* para a toolchain do NAO e então se poderá compilar o arquivo fonte do ROS, que terá a inclusão da toolchain do NAO e irá gerar o projeto. O passo seguinte será copiar os pacotes *yaml*, *rospkg*, o script *pkg_resources* e as bibliotecas *liblog4cxx, libapr* e *libboost* para o NAO (lembrando de especificar seu path para que o ROS os reconheça). Finalmente deverá se copiar, via ssh, a pasta de instalação para o diretório raiz do robô. Quando feita a instalação é possível testa-la utilizando o comando roscore[2].

B. Estrutura do sistema

Esta seção será dedicada a explicação dos diversos nós que compõe o sistema, mostrados no diagrama da Fig. 1.

1) Comunicação: Na SPL, como visto na Subseção I-B, utiliza-se o Game Controller para controlar a partida, tornando fundamental sua comunicação com o NAO. Logo, para possibilitar a interpretação das mensagens recebidas e informar ao robô o estado atual do jogo foi criado o nó de comunicação. Este módulo se comunica com o Game Controller e obtêm as informações referentes ao número

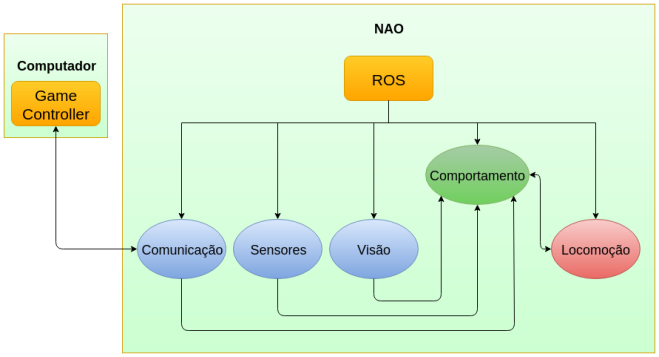


Fig. 1: Diagrama de funcionamento do sistema integrando o NAO e o ROS

[1]http://doc.aldebaran.com/1-14/
[2]http://wiki.ros.org/nao/Installation/Cross-Compiling_NAO-V4

do time, número do jogador e cor do time, além do valor de estado atual de jogo e se o robô está penalizado. A comunicação dá-se por meio de protocolos UDP e após a obtenção dos dados desejados estes são publicados na forma de serviços. Foi, também implementada uma rotina utilizando LEDs como ferramenta de debugging, assim cada estado acionado pelo Game Controller deveria gerar uma cor diferente no peito do robô.

2) Locomoção: Com o intuito de implementar as diversas rotinas de movimentação, foi criado o nó *Motion*. Como mostrado pela Fig. 2. Para a implementação deste nó foram utilizadas as bibliotecas padrão do NAO, contidas no framework disponibilizado pela Aldebaran, o Naoqi.

Inicialmente implementou-se as funções responsáveis pela marcha do robô: *MoveTo*, *MoveAroundPoint*, *Rotate*, *MoveTowards* e *StopMove*. Estas funções são definidas pela chamada de rotinas de movimento equivalentes às disponíveis na Naoqi, além de simples algoritmos de transformações de coordenadas. As funções *MoveTo* e *Rotate* descrevem a trajetória do robô em termos de coordenadas cartesianas. A *MoveAroundPoint* faz com que o robô descreva uma trajetória circular em torno de um ponto qualquer, como a bola. A função *MoveTowards* descreve uma trajetória baseada em parâmetros de velocidade e, por fim, a função *StopMove* faz com que o robô pare.

Quanto as funções relativas a atuação dos motores para realização de ações, tem-se a *MoveJoint*, que move uma junta, recebendo como referência o ângulo desejado e a velocidade do movimento. Tem-se também a função *GoToPosture*, responsável por mover o robô para uma postura pré definida, e a função *StopPosture* que simplesmente encerra a movimentação do robô.

As funções *KickLeft* e *KickRight* promovem o chute na bola, sendo implementadas através da aplicação Choreographe, da Aldebaran Robotics. Com esta aplicação é possível modelar uma ação manualmente e extrair os valores referentes as posições das juntas em alguns pontos no tempo. Desse modo, utilizando a função *angleInterpolationBezier*, interpola-se a curva de Bézier e realiza-se a atuação de um movimento fluido.

Quanto as outras funções tem-se o *StiffnessOn* e *StiffnessOff,* que definem a rigidez para as juntas do robô, e a função *GetJointAngle*, que retorna o ângulo em radianos de uma junta especificada.

3) Sensores : O nó responsável pela interface com os sensores foi implementado com os serviços mostrados na

Motion

Andar	Ação	Outros
Move To	Move Joint	Stiffness On
Move Around Point	Go To Posture	Stiffnes Off
Rotate	Stop Posture	Get Joint Angle
Move Toward	Kick Right	
StopMove	Kick Left	

Fig. 2: Serviços publicados pela função *Motion*

Fig. 3. O NAO possuí diversos sensores integrados em sua arquitetura. No torso encontram-se dois sonares, um par com receptor e um transmissor a esquerda e outro a direita, utilizados na detecção de obstáculos. Localizada, também no torso do robô tem-se a unidade inercial, que é composta por um giroscópio de dois eixos e um acelerômetro de três. Além destes, o NAO possuí também dois sensores infra-vermelho nos olhos, quatro sensores de pressão em cada pé, sensores capacitivos na cabeça e em cada mão, um botão no torso, um bumper em cada pé e encoders que fornecem a posição de cada junta.

Com os diversos sensores disponíveis criou-se um módulo de acesso à memória do robô, sendo elaboradas funções para a aquisição dos sensores da unidade inercial (acelerômetro e giroscópio) e aos sonares. Essas informações são publicadas diretamente na memória compartilhada do robô e são acessadas com a biblioteca padrão Naoqi.

No caso dos sensores da unidade inercial tem-se as funções *GetAccel*, referente ao acelerômetro, que retorna os valores da aceleração (em m/s^2) nos eixos cartesianos referentes ao robô. Existe também a *GetGyro*, que se refere ao giroscópio, retornado a velocidade angular de rotação do robô (pitch e yaw). A *GetIMU*, utiliza os dados do acelerômetro e giroscópio para calcular o angulo do robô em radianos.

Os outros serviços são referentes a função *GetSonar*, que retorna a distância em metros de um objeto localizado por algum dos sonares, e a função *RobotHasFallen*, que retorna um valor booleano que indicando se o robô caiu.

4) Visão: O nó da visão é responsável por adquirir as imagens de ambas as câmeras do robô. A partir dessas imagens são processadas informações referentes ao histograma, rastreamento de bola e detecção do gol. Os códigos desenvolvidos utilizam ferramentas da biblioteca OpenCV.

A função de rastreamento da bola se dá inicialmente pela aplicação de um filtro gaussiano e pela segmentação da imagem para a cor laranja, então são feitas transformações morfológicas de dilatação e erosão para diminuir o ruído. Em seguida são encontrados os momentos da imagem, e a partir destes, são encontrados os centros de massa do objeto e suas coordenadas. Em seguida utiliza-se a função *findContours*, do OpenCV, para se obter um vetor de contornos e, subsequentemente, calcular a área de cada elemento, sendo que áreas muito pequenas serão descartadas. Finalmente, para verificar se a bola está na imagem é usada uma função que checa se os pontos do centro de massa se encontram dentro dos polígonos formados pelos contornos: em caso positivo, a bola foi encontrada.

Também tem-se a função em que se determina o histograma, usada para identificação do campo adversário. A implementação dessa função ocorre a partir da comparação de cada canal de uma foto tirada no início da partida com os canais equivalentes de uma imagem extraída da câmera no momento do chute. Tira-se, a média dos resultados do histograma e caso este seja maior que um limiar (60% para os experimentos realizados) considera que as imagens são iguais e o robô estará olhando para o lado adversário.

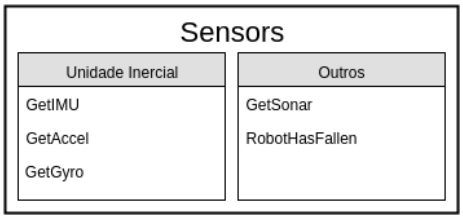


Fig. 3: Serviços que publicados pela função *Sensors*

5) Comportamento: Como visto no diagrama do sistema mostrado na Fig. 1, a rotina de comportamento recebe os serviços de todos os outros nós que estão rodando em paralelo. Com os dados recebidos será implementada uma maquina de estados que, primeiramente, analisará o estado do jogo a partir das informações fornecidas pelo módulo de comunicação. Os possíveis estados são *Initial*, *Ready*, *Set*, *Playing* e *Penalized*, como mostrado na Fig. 4. Caso não haja comunicação com o Game Controller os estados serão definidos através do botão localizado no torso do robô.

O estado *Initial* é responsável por definir a postura do robô e tirar a foto inicial que será utilizada para calcular o histograma. O estado seguinte é o *Ready*, responsável por definir parâmetros iniciais de odometria e realizar o posicionado do robô próximo ao centro do campo. Caso a comunicação não seja exercida pelo Game Controller, o robô será colocado manualmente no centro do campo. Durante o estado *Set*, não deverá haver se deslocamento, somente a movimentação da cabeça e o rastreamento da bola. O estado *Playing* gerencia todo o comportamento durante o jogo. Por fim, o estado *Penalized* encerra todas as funções do robô até que este receba algum comando para regressar ao estado anterior.

Dentro do módulo de comportamento o mais importante é o estado *Playing*, mostrado na Fig. 5. Neste o robô deverá inicialmente procurar a bola pelo campo. Cabe lembrar que a bola pode estar parcialmente oculta ou em movimento, optando-se então por uma estratégia de buffer para realizar seu rastreamento. O buffer guarda em sua memória as últimas três posições em que a bola foi encontrada e utiliza essa informação para determinar se ela realmente está naquele local. Em caso positivo o robô se moverá na direção desejada. A movimentação do robô em direção a bola possui um erro:

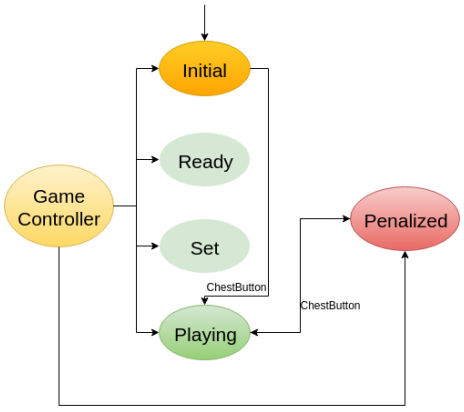


Fig. 4: Estados Fornecidos pelo Game Controller

trajetórias que deveriam ser retas se tornam curvas. Para corrigir este problema, foi utilizado um controle proporcional integral. O estado seguinte deverá ser acionado quando a bola for encontrada na câmera inferior e estiver próxima o bastante para que haja um chute, o robô deverá alinhar-se com a bola e preparar-se para chutar. Antes do chute, uma foto será tirada e seu histograma será calculado, a fim de fazer uma comparação com o encontrado no início do jogo. Caso o histograma seja coerente, o NAO deverá entrar em estado de chute. Caso contrário, deverá contornar a bola e orientar-se para o lado oposto. Ainda no estado de chute, o robô deverá definir o pé mais eficiente para realizar a ação. Por ultimo, ele voltará para o estado de procurar a bola.

Em paralelo com a busca acontece a leitura do sonar, que ocorre em todas as iterações do módulo. Caso a leitura retorne como positiva, o robô deverá entrar em um estado de desvio, para evitar colisões com qualquer objeto no campo.

III. Resultados e Análise

No âmbito experimental, os módulos foram verificados individualmente em laboratório e testados durante duas competições: a RoboCup 2014 e a LARC 2014. Com base no observado e se embasando nos arquivos de vídeo, realizou-se a análise qualitativa do funcionamento do sistema.

A. Avaliação da Comunicação

O nó responsável pela comunicação tem como principal objetivo interagir com o Game Controller. O robô deveria receber os estados no formato protocolado e a partir destas informações realizar as ações correspondentes.

Durante o desenvolvimento em ROS foram utilizados LEDs como ferramenta de debugging, sendo assim possível observar o comportamento do robô em relação as respostas das mensagens. De fato, foi visto que com as chamadas dos diferentes estados do jogo os olhos do robô tomavam corretamente as cores designadas. Além disso, cada estado do jogo era programado com um comportamento próprio, sendo assim possível observar, com a atuação correta do robô, que as chamadas ocorriam na hora devida.

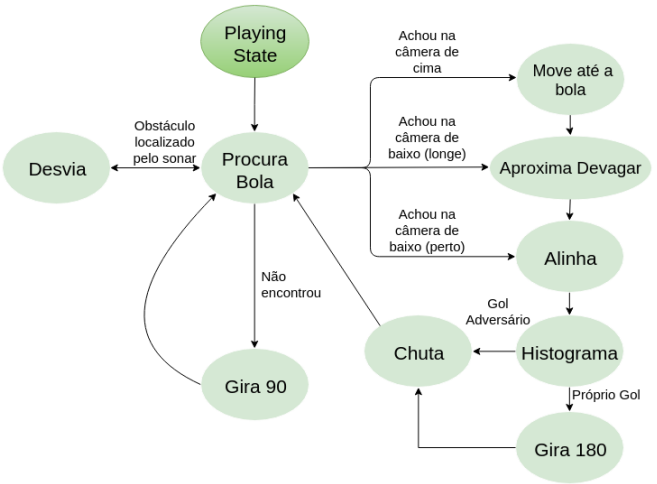


Fig. 5: Máquina de estados dentro do *Playing*

Tendo em vista o sucesso obtido na comunicação com o Game Controller e observando as dificuldades apresentadas por muitas equipes da RoboCup em estabelecer este protocolo, pode-se considerar a implementação deste módulo efetiva.

B. Avaliação da Locomoção

Este módulo foi testado a partir da observação dos movimentos, que de fato foram executados quando requerido. Visto que muito do que foi implementado se deu por uma chamada direta das funções presentes na Naoqi, observou-se efetividade do movimento. Em relação aos movimentos criados pelo time, como o chute, também se observou efetividade, sendo que mesmo sem uma realimentação para garantir correções de erro (o que teria um custo alto no sistema) o robô se manteve estável, além de conseguir garantir um bom alcance para a bola.

Ao analisar a movimentação do robô e compará-la com a de outras equipes da SPL pode-se perceber uma grande diferença entre as implementações. Há a preferência dentro da comunidade de competidores por rotinas de movimento diferentes das disponibilizadas pela Aldebaran [7], [8]. As implementações feitas por essas equipes tenderam a valorizar a performance em termos de velocidade, em detrimento a estabilidade. Movimentos como levantar, chutar, além da própria marcha do robô são, de fato, mais rápidos do que os movimentos usados nesse sistema. No entanto, apesar da importância desses aspectos em um ambiente competitivo, notou-se também que é imprescindível que o robô se mantenha estável. Quando feita uma comparação em termos de estabilidade dos movimentos, foi claro que o robô cujo sistema era baseado em funções da Aldebaran caía e escorregava bem menos.

C. Avaliação dos Sensores

Nesse módulo esperava-se receber corretamente os dados dos sensores, sendo este testado por meio de uma análise da resposta dos robôs a certos eventos.

Primeiramente temos os dados da unidade inercial e o evento *RobotHasFallen*, que indicam quando o robô caiu. Foi possível verificar o funcionamento correto destas funções quando o robô foi colocado na posição horizontal e iniciou o processo de se levantar. A outra função utilizada nesse módulo é a de obtenção dos dados relativos ao sonar. Este foi testado com o comportamento de desvio em que a detecção de cada sonar resultava no movimento para o lado oposto. De fato, o robô sempre parava e em seguida andava para o lado desejado: para a esquerda quando desviando de objetos à direita; para a direita quando detectados objetos à esquerda; e para a direita quando um objeto era detectado a frente pelos dois sonares simultaneamente.

D. Avaliação da Visão

Primeiramente foi feita a análise da função de rastreamento da bola, em que foi observada a eficácia do algortimo implementado. Foi evidente que toda vez que a bola se

encontrava no campo de visão do robô este deslocava-se até ela, havendo somente uma dependência em relação aos valores de calibração.Nesse aspecto, foi criada uma aplicação que se conecta com a câmera do robô e permite que o usuário mude os parâmetros de calibração até encontrar valores satisfatórios. Apesar desta sofrer interferência humana e haver um erro experimental, a calibração tende ser confiável.

O módulo referente ao histograma, tem o objetivo de identificar o lado adversário, para onde o robô deverá chutar. Esse método se provou eficiente durante os experimentos realizados na RoboCup 2014, como pode ser observado na Fig. 6, em que pode-se notar uma grande diferença no padrão entre um lado e ou outro do campo. Entretanto, durante os outros experimentos realizados foi verificado, que ambos os lados possuíam condições similares de cor e iluminação, ocasionando diversos erros na determinação do campo adversário.

E. Avaliação do Comportamento

Ao se analisar o nó de comportamento, pode-se verificar o sistema como um todo. As rotinas implementadas mostraram resultados efetivos, sendo que a maioria dos problemas encontrados foram provenientes de dados falhos vindo de outras partes do sistema. No âmbito do comportamento em si, todos os estados funcionavam.

De fato, o robô respondeu aos estados do Game Controller quando requisitado e na falta deste seguiu os estados definidos pelo botão. Quando no estado *Playing*, pode-se notar que o robô estava seguindo a máquina de estados da Fig. 5.

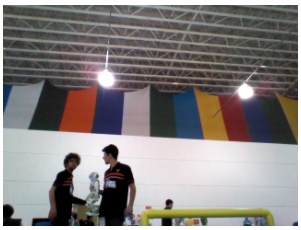
Durante as experiências feitas optou-se por uma estratégia mais conservadora e defensiva, na qual se priorizava evitar obstáculos. Ficou evidente, ao se observar outros times, que a maioria dos robôs tinha como máxima prioridade correr atrás da bola, desse modo todos eles tendiam para o mesmo ponto e se chocavam, o que era muitas vezes ineficaz.

A estratégia defensiva se mostrou uma boa escolha, levando o robô a tomar decisões melhores, além de preservar sua integridade física.

IV. Conclusão e trabalhos futuros

Ao analisar-se os experimentos, é evidente que a implementação do sistema no framework ROS é vantajosa tendo em vista a sua fácil integração com as bibliotecas utilizadas, além de possibilitar um gerenciamento seguro entre os serviços. Entretanto, apesar dos resultados positivos obtidos até o momento, o sistema ainda encontra-se incompleto, sendo necessária a ampliação do número de nós e implementação de melhorias nos já existentes.

(a) Campo adversário



(b) Campo defendido

Fig. 6: Diferença entre os lados do campo na RoboCup 2014

Em trabalhos futuros deverão ser desenvolvidas rotinas de navegação e planejamento de rotas, identificação de outros robôs, protocolos de comunicação entre jogadores, além da resolução dos diversos desafios que são propostos anualmente nos eventos. No mais, nós como Visão e Locomoção podem ter suas funcionalidades expandidas e seu desempenho melhorado com a implementação de novos algoritmos.

Entretanto, é importante ressaltar que a arquitetura proposta, apesar de robusta, apresentou inconvenientes quanto a sua velocidade de execução. O próprio processo do ROS, em combinação com os vários nós rodando em paralelo, além de alguns métodos com códigos desenvolvidos em uma linguagem interpretada, o Python, resultaram em uma restrição da frequência com que os nós do sistema podem ser executados. Levando em conta os requisitos de uma aplicação voltada para competições como a RoboCup e LARC, em que diversos processos de alto custo computacional devem ser executados concomitantemente, foi observada imcompatibilidade da arquitetura proposta com o hardware disponível no NAO, sobretudo se considerados os futuros nós previstos para a expansão do sistema, com rotinas de mais alto nível.

Conclui-se, portanto, que o uso do ROS na plataforma NAO é mais indicado para aplicações com número reduzido de processos simultâneos ou que não contenha fortes restrições de tempo de execução. Nos casos em que não é possível reduzir a quantidade de módulos ou flexibilizar os requisitos de velocidade computacional, torna-se necessário substituir o ROS por uma arquitetura mais otimizada, ou utilizar um hardware mais potente.

Referências

[1] H. Feng, A. Gutierrez, J. Zhang, and M. Mahoor, "Can NAO robot improve eye-gaze attention of children with high functioning autism?," in *Healthcare Informatics (ICHI), IEEE Int. Conf. on*, p. 484, 2013.

[2] A. Alkhalifah, B. Alsalman, D. Alnuhait, O. Meldah, S. Aloud, H. Al-Khalifa, and H. Al-Otaibi, "Using NAO humanoid robot in kindergarten: A proposed system," in *Advanced Learning Technologies (ICALT), IEEE 15th Int. Conf. on*, pp. 166–167, July 2015.

[3] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[4] M. P. de Carvalho, "Controle de movimentação de humanoide em tempo real por teleoperação," trabalho de conclusão de curso em eng. mecatrônica, Univ. de Brasília, 2014.

[5] F. M. Ramos, "Aplicação de raciocínio formal para percepção e manipulação de objetos com o robô humanoide NAO," trabalho de conclusão de curso em eng. mecatrônica, Univ. de Brasília, 2015.

[6] L. L. Forero, J. M. Yánez, and J. Ruiz-del Solar, "Integration of the ROS framework in soccer robotics: the NAO case," in *RoboCup 2013: Robot World Cup XVII*, pp. 664–671, Springer, 2013.

[7] T. Röfer, T. Laue, J. Richter-Klug, M. Schünemann, J. Stiensmeier, A. Stolpmann, A. Stöwing, and F. Thielke, "B-Human team report and code release 2015." `http://www.b-human.de/downloads/publications/2015/CodeRelease2015.pdf`, 2015.

[8] B. Hall, S. Harris, B. Hengst, R. Liu, K. Ng, M. Pagnucco, L. Pearson, C. Sammut, and P. Schmidt, "RoboCup SPL 2015 champion team paper." `http://www.cse.unsw.edu.au/opencms/export/sites/cse/about-us/help-resources/for-students/student-projects/robocup/reports/SPL2015ChampionTeamPaper.pdf`, 2015.