XXXV CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO
De 20 de Julho a 23 de Julho de 2015
Recife - PE

Anais da 34ª Jornada de Atualização em Informática

JAI 2015

Editora
Sociedade Brasileira de Computação (SBC)

Realização
Centro de Informática (CIn) da UFPE

Promoção
Sociedade Brasileira de Computação (SBC)

Sumário

Capítulo

6

Simulação de Robôs Móveis e Articulados: Aplicações e Prática

Rafael Alceste Berri, Valdir Grassi Jr. e Fernando Santos Osório

Abstract

This course aims to provide an introduction to key concepts of mobile robotics and articulated robots, through simulations and based on practical examples presented in the course. Practices will be made available to the participants. The R&D of Intelligent Mobile Robots is an important academic and industrial field nowadays, which is directly related to mobile robots design, autonomous vehicles, humanoid robots, industrial manipulator arms and aerial robots. The tool we adopted in this short-course allows simulating different types of mobile and articulated robots, as well as, simulates sensors and other elements of the virtual environment. In this short-course different concepts about mobile and intelligent robots will be addressed: sensors, actuators, robot configuration (kinematic models), and intelligent robot behavior programming, that is, the implementation of behaviors integrating the robot perception-decision-action cycle. This simulator tool, V-REP PRO (Free and Non-Limited Educational Version), lets the user play with realistic situations (high degree of 3D realism including physical simulation), and also allows testing different models of well-known robots and sensors. This short-course will provide a direct contact with robotics and will open several different opportunities for new studies and research development in this field, once the simulator reproduces quite closely real-world robotic applications.

Resumo

Este curso visa apresentar uma introdução aos principais conceitos da área de robótica móvel e sobre robôs articulados, através de uma abordagem baseada em simulações e orientada a exemplos práticos apresentados no curso, os quais serão disponibilizados e poderão ser testados pelos participantes. A robótica móvel inteligente é uma importante área em grande expansão na atualidade, onde está diretamente relacionada ao projeto de robôs móveis, veículos autônomos, robôs humanoides, braços manipuladores e robôs

aéreos. A ferramenta adotada neste mini-curso permite simular diferentes tipos de robôs móveis e articulados, além de sensores e outros elementos que podem compor o ambiente virtual de simulação montado pelo usuário. Neste mini-curso serão abordados os conceitos sobre sensores, atuadores, configuração de robôs (modelos cinemáticos), e sobre a programação de comportamentos inteligentes, ou seja, a implementação de comportamentos integrando a percepção-decisão-ação do robô. O simulador adotado permite reproduzir situações reais com um alto grau de realismo 3D, além de permitir a realização de testes com variados modelos de robôs e de sensores. O curso provê um contato direto com a robótica e com inúmeras possibilidades de desenvolvimentos de estudos e pesquisas, usando um simulador que reproduz de modo bastante fiel as aplicações robóticas do mundo real.

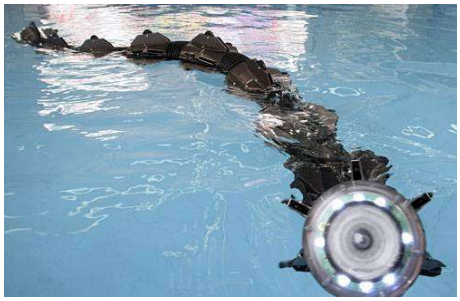
6.1. Introdução

A palavra robô é originária da palavra tcheca *robota* que significa literalmente "trabalhador humilde" [Romero et al. 2014a]. Sua primeira aparição foi em uma peça teatral de Karel Capek intitulada R.U.R.(*Rossumś Universal Robots*), de 1921, onde Rossum, um inventor, cria trabalhadores artificiais (os *roboti*), aptos a desempenhar algumas tarefas humanas do quotidiano.

Da ficção até os dias atuais, a robótica tem avançado significativamente. Inúmeros novos recursos de *hardware* e *software* vem aparecendo e facilitando a sua ampla disseminação em nossa sociedade, tornando-se cada vez mais difícil desvincular os robôs do futuro da humanidade. Em 2007, Bill Gates em seu artigo publicado na *Scientific American* [Gates 2007], comparou o crescimento da robótica com a disseminação do computador nos lares, onde faz a alusão de que "PCs irão deixar o seu lugar em cima das mesas para passar a ver, ouvir, tocar e manipular objetos".

Os robôs móveis (com capacidade de locomoção pelo ambiente) e robôs articulados (possuem uma base fixa), sejam eles projetados para executar tarefas repetitivas ("não inteligentes"), para serem teleoperados, ou para operar de forma autônoma e inteligente, constituem uma importante área de pesquisa e desenvolvimento em grande expansão na atualidade [Department 2014]. O desenvolvimento de pesquisas e projetos de robôs móveis tem resultado em produtos e empresas com grande crescimento. Este avanço robótico já se apresenta a disponibilidade de robôs nas mais diversas formas e tipos. Na Figura 6.1 são mostrados exemplos de alguns robôs, onde, o ACM-R5 [HiBot 2015] da *HiBot* (Figura 6.1(a)) é um robô serpente que consegue operar em baixo da água, sendo já utilizado em filmes de *Holywood*; o humanóide NAO [Aldebaran Robotics 2015] (Figura 6.1(b)) desenvolvido pela Aldebaran Robotics e o Pioneer p3dx [Adept MobileRobots 2015] (Figura 6.1(c)) são grandemente utilizados em pesquisas científicas; o braço robótico IRB 140 [ABB 2015a] (Figura 6.1(d)) é para uso industrial da ABB; tratando-se de ambientes domésticos, o Roomba [iRobot 2015] (Figura 6.1(e)) da iRobot é um aspirador de pó robótico e o Landroid Wg794e [Landroid 2015] (Figura 6.1(f)) é um cortador de grama autônomo; e, mencionando robô de muitas utilidades no espaço, pode-se citar o Robonaut 2 (Figura 6.1(g)) e os *rovers* de exploração de Marte da Nasa [NASA 2015].

Dentro deste contexto, é de grande impacto econômico e de grande relevância social e industrial, o desenvolvimento da robótica no Brasil, para permitir que mais pessoas



(a) ACM-R5

(b) NAO

(c) Pioneer p3dx

(d) IRB 140

(e) Roomba

(f) Landroid Wg794e

(g) Robonaut 2

Figura 6.1. Exemplos de Robôs: o (a) ACM-R5 [HiBot 2015], o (b) NAO [Aldebaran Robotics 2015] e o (c) Pioneer p3dx [Adept MobileRobots 2015], (d) IRB 140 [ABB 2015a], o (e) Roomba [iRobot 2015], (f) Landroid Wg794e [Landroid 2015] e o (g) Robonaut 2 [NASA 2015].

possam fazer parte dessa revolução. As pesquisas e desenvolvimentos em robótica envolvem diversas áreas, tais como: Computação (*software*), Engenharia Eletrônica (*hardware*) e Mecânica (projeto da estrutura do robô), entre outras áreas de conhecimento (p.ex. Física, Matemática). A computação tem um papel muito importante, pois provê o suporte necessário para possibilitar a criação de sistemas de controle mais robustos, autônomos, inteligentes e seguros para os robôs.

A adoção de *softwares* de simulação robótica realista tem um papel extremamente benéfico ao acesso à robótica. Pois torna possível recriar virtualmente um determinado robô (*hardware*) em um mundo virtual, e assim, desenvolver e preparar o software reduzindo custos, tempos de desenvolvimento e experimentação, ou riscos de dano a um robô real.

É de grande importância na atualidade à formação de pessoas capacitadas nesta área, com conhecimentos sobre o tema, e capacitadas para o projeto e desenvolvimento de aplicações robóticas. Neste curso serão abordados os conceitos sobre sensores, atuadores, configuração de robôs (modelos cinemáticos), e sobre a programação de comportamentos inteligentes, ou seja, a implementação de comportamentos integrando a percepção-decisão-ação do robô. O simulador adotado neste curso, o V-REP [E. Rohmer 2013] [1], permite a reprodução de situações reais com um alto grau de realismo (3D, simulação física), além de permitir a realização de testes com os mais variados modelos de robôs (por exemplo Pioneer, NAO Humanóide, Braços Robóticos, Robôs articulados com patas, robôs aéreos) e de sensores (Laser Sick e Hokuyo, Câmeras, GPS, Kinect, Velodyne). Este simulador é de uso livre para fins acadêmicos, permitindo uma fácil e ampla configuração e programação dos robôs. O uso de um simulador realista e flexível facilita assim um contato direto com a robótica e com as inúmeras possibilidades de desenvolvimentos de estudos e pesquisas nesta área [2].

6.2. Conceitos sobre Robôs Móveis e Robôs Articulados

Robôs podem possuir dois tipos de bases distintas. Os robôs de base fixa são aqueles que permanecem fixados a um local específico, como por exemplo, os braços robóticos industriais, e possuem articulações (por isso são denominados de Robôs Articulados ou Braços Manipuladores) que permitem efetuar o trabalho para o qual foram projetados. Já os de base móvel possuem a capacidade de se locomover pelo ambiente em que se encontram, esses robôs são chamados de Robôs Móveis por essa razão. Os robôs móveis podem ainda ser tele-operados, semi-autônomos, autônomos ou dotados de sistemas inteligentes de tomada de decisão, constituindo os denominados veículos autônomos inteligentes [Jung et al. 2005]. Portanto, estes últimos possuem a capacidade de locomoção e de operação semi ou completamente autônoma, atuando em ambientes totalmente controlados (ou não) e bem estruturados (ou não).

Os robôs são usualmente compostos por diversos dispositivos e módulos, destacando-se os sensores (capacidade de percepção do ambiente onde ele atua), o sistema de processamento de informações (planejamento, decisão e controle), e os atuadores (geram as ações, através de motores capazes de produzir estas ações). Nas próximas subseções

[1] V-Rep Site: http://www.coppeliarobotics.com/

[2] Site com informações complementares e demos do V-REP: https://www.sites.google.com/site/vrepjai/

são mostrados alguns exemplos dos diversos sensores (Seção 6.2.1) disponíveis; em seguida os elementos usados para dar mobilidade e movimento as articulações dos robôs, os atuadores (Seção 6.2.2); na Seção 6.2.3 serão discutidos os diferentes tipos de atuadores e como estes podem afetar o comportamento e movimentação do robô de acordo com sua utilização; na sequência as arquiteturas de controle (Seção 6.2.4), ou seja, os modelos computacionais adotados para implementar o sistema de planejamento, decisão e ação de robôs são mostrados; e então na Seção 6.2.5 é apresentada uma discussão sobre os comportamentos mais simples adotados em sistemas robóticos móveis/articulados. Estes comportamentos é que nos permitem implementar tarefas robóticas mais sofisticadas (p.ex. mapeamento do ambiente, determinação da localização do robô em mapas, navegação até uma determinada posição, desvio de obstáculos), envolvendo a percepção, decisão e ação dos robôs.

6.2.1. Sensores

Sensores podem ser chamados também de transdutores já que a ideia por de trás deste equipamento é transformar a energia medida em outro formato de apresentação, mais simples para a utilização [Murphy 2000]. Por exemplo, um sensor pode captar o som, a pressão ou mesmo a luz, convertendo em um sinal analógico ou digital que possa ser usado pelo robô.

Existem diversos tipos de sensores [Wolf et al. 2009], porém podemos classificar em dois tipos principais segundo seu método de medição: os sensores ativos emitem energia no ambiente e mensuram seu retorno; e os sensores passivos somente captam a energia já disponível no ambiente [Siegwart et al. 2011].

A tarefa do robô deve ser levada em conta no momento da escolha do sensor, já que cada sensor possui características próprias, como por exemplo, faixa de atuação, sensibilidade, precisão e exatidão [Romero et al. 2014b]. Quando não existem disponíveis sensores com características compatíveis com a tarefa, uma alternativa adotada é a fusão de múltiplos sensores. A fusão sensorial pode ser classificada em [Romero et al. 2014b]: (*i*) redundante ou competitiva, onde, mais de um sensor com características iguais ou distintas fazem a medições do ambiente para posterior confronto (normalmente utilizada quando o problema a ser tratado é a imprecisão); (*ii*) complementar, sensores que medem informações distintas e que juntos são capazes de eliminar falsas detecções; e (*iii*) coordenada na qual se utiliza sensores em uma ordem específica, como por exemplo, sendo detectado som em alguma sala, então outro tipo de sensor é utilizado para se confirmar mais informações, como uma câmera.

Os sonares são sensores ativos capazes de detectar a distância entre o sensor e os objetos/obstáculos. A detecção ocorre pela emissão de um pacote sonoro ultrassônico, onde, mede-se o tempo em que o som demora a refletir em alguma superfície e retornar. Os sonares são equipamentos usualmente de baixo custo e possuem um alcance curto, detectando normalmente objetos entre 30 centímetros e 5 metros [Romero et al. 2014b]. A precisão da medição também depende do ângulo em que as ondas atingem uma superfície, quanto mais agudo maior é o erro da medição. Um robô pode utilizar vários sonares, porém precisam ser acionados de modo sequencial para minimizar eventuais interferências entre eles. Na Figura 6.2(a), mostra-se um exemplo de sonar o HC-SR04 da Cytron

Technologies.

Os sensores a *laser* são denominados de LIDAR (*Light Detection And Ranging*) e detectam distâncias de maneira similar ao cálculo feito pelo sonar, mas utilizando o tempo entre a emissão e recepção de pulsos de *laser*. A luz possui velocidade de propagação mais rápida que o som e melhor direcionada (no caso do laser), por isso os sensores LIDARs podem ter um ciclo de execução mais curtos e melhor precisão na estimativa de distâncias. Dois exemplos de sensores LIDAR utilizados em robótica são o sensor Hokuyo URG-04LX-UG01 [Hokuyo Automatic 2015] (Figura 6.2(b)) e o Sick LMS 200 [SICK 2015] (Figura 6.2(c)). O Hokuyo URG-04 é um sensor possui um alcance de 4 metros e um campo de visão de 240°. O Sick LMS-200 possui alcance de 10 metros (em condições ideais chega a 80 metros), um campo de visão de 180° e um custo significativamente maior que o Hokuyo para aquisição. Ambos os sensores redirecionam o feixe de *laser* usando um espelho [3], permitindo assim, a varredura do ambiente. Existem alguns fatores que podem dificultar a medição das distâncias [Romero et al. 2014b]: objetos escuros (a cor preta absorve a luz/*laser*), quanto mais distantes os objetos mais impreciso é a leitura (menos luz retorna ao sensor) e a incidência do *laser* sobre superfícies com propriedades específicas de reflexão/refração/absorção de luz (ex: espelhos e vidros) causam falhas de leitura.

Câmeras de vídeo são sensores comumente aplicados em projetos robóticos, apresentando uma rica quantidade de informações sobre a área de atuação visualizada pelo robô [Romero et al. 2014b] [Mataric 2014]. Com este ganho de informação é possível dotar o robô de um sistema de percepção similar ao humano, obtendo distâncias (como sensores laser e ultrassom), presença, postura, gestos, etc. As câmeras atuais capturam a informação utilizando a tecnologia CCD ou CMOS. O CCD (*charged coupled device*) possui uma matriz sensível à luz (fotodiodo) ou *pixels*. A energia projetada sobre o *pixel* é acumulada por certo tempo, sendo então congelada e transmitida a um registrador. O registrador processa uma linha da matriz de pixels por vez, amplificando o sinal de cada *pixel* e convertendo para digital, gerando assim a imagem final. A Figura 6.2(d) apresenta a câmera Point Grey Chameleon[Point Grey Research 2015] que utiliza matriz CCD. A tecnologia CMOS (*Complementary Metal-Oxide Semiconductor*) possui uma matriz de pixel similar ao CCD, mas tendo como grande diferença, a existência em cada *pixel*, de um registrador dedicado, que amplifica e digitaliza o sinal capturado. Não necessitando, portanto, de congelamento ou transmissão de informações para um registrador externo. As duas tecnologias vem competindo pela preferência dos fabricantes de câmeras a alguns anos, mas o que deve ser levado em conta no momento de se optar por uma ou por outra é que a CCD possui maior sensibilidade a luz e menor apresentação de ruídos; já o CMOS pode trabalhar com uma frequência de captura mais elevada, possui menor custo de fabricação e menor consumo de energia [Siegwart et al. 2011]. CCD, portanto, possui uma qualidade de imagem melhor, mas CMOS vem apresentando significativas melhorias e já é possível encontrar câmeras CMOS com qualidade similar a CCD [Romero et al. 2014b].

As câmeras de vídeo permitem a aquisição de imagens, que são mais ricas em

[3]LIDAR - Animação do funcionamento: http://upload.wikimedia.org/wikipedia/commons/c/c0/LIDAR-scanned-SICK-LMS-animation.gif

termos de informações, porém usualmente mais complexas de serem tratadas. Os sistemas de visão computacional buscam explorar as informações capturadas pelas câmeras e assim contribuir na percepção do ambiente, seja usando imagens estáticas ou através de sequências de imagens (vídeos). As câmeras permitem identificar elementos pela sua forma e contornos, cor, textura e até mesmo estimar a proximidade de obstáculos.

Um sensor de posicionamento muito usado na robótica (para ambientes externos) é o *Global Positioning System* (GPS) ou Sistema de Posicionamento Global. O GPS utiliza-se dos satélites de posicionamento que estão na órbita da terra. Esses satélites emitem mensagens contendo a hora de partida da mensagem. Assim, o GPS calcula a distância que a mensagem percorreu para chegar e, tendo recebido mensagens de pelo menos 2 satélites [Siegwart et al. 2011], é possível estimar a posição do sensor na superfície da terra. O GPS é um sensor absoluto, ou seja, cada cálculo de posicionamento é independente do anterior, portanto, seu erro é local e não cumulativo [Romero et al. 2014b].

O giroscópio é um dispositivo capaz de informar a direção para onde está se movendo. Isso faz com que seja usado para auxiliar em navegação de helicópteros e aviões (principalmente no piloto automático). Existem dois tipos de giroscópios, os mecânicos e os óticos [Siegwart et al. 2011]. Os mecânicos são baseados no princípio da inércia, usando um rotor suspenso por um suporte formado por dois círculos articulados (um exemplo de giroscópio mecânico é mostrado na Figura 6.2(e)). Os ópticos medem a velocidade angular entre dois feixes de luz ou *lasers* emitidos de uma mesma fonte. O erro de medição do giroscópio é cumulativo, ou seja, erros anteriores (ou perdas de leituras de dados) atrapalham nas medições futuras e estimativas de posicionamento e orientação.

Um acelerômetro também é uma unidade inercial como o giroscópio, porém, mede a ação de forças externas sobre ele, inclusive a gravidade. Atualmente, os acelerômetros são pequenos sistemas Microeletromecânicos (MEMS) que através de uma pequena coluna (massa) balançante, as forças são mensuradas. Um exemplo de acelerômetro é o MMA7361L [Freescale Semiconductor 2015], mostrado na Figura 6.2(f). Os acelerômetros, assim como o giroscópio, também sofrem do problema do erro cumulativo, que pode levar a erros em estimativas de posição e orientação. Por isso muitas vezes são aplicados filtros para eliminar ruídos e melhorar as estimativas, como o Filtro de Kalman [Wolf et al. 2009].

Existem dispositivos que agregam um giroscópio e um acelerômetro internos, podendo até fazer fusão de mais sensores, como bússola. Assim são capazes de estimar a posição relativa, velocidade e aceleração do movimento de um veículo ou robô. Esses dispositivos integrados são chamados de IMUs (*Inertial Measurement Units*), porém carregam os mesmos erros cumulativos do giroscópio e acelerômetro. É possível, no entanto, adotar alguma referência externa como um GPS para minimizar esse problema, bem como aplicar filtros. Um exemplo de IMU é o MPU6050 [InvenSense 2015] mostrado na Figura 6.2(g).

A profundidade do ambiente pode ser observada através de sensores 3D, como por exemplo, a câmera estéreo. Ao se utilizar algoritmos de processamento das imagens proveniente de duas câmeras defasadas (dois pontos de vistas diferentes, normalmente próximos) podem ser gerados mapas de disparidade [Bradski and Kaehler 2008], onde

a relação entre o mesmo pronto visto de diferentes posições (por cada uma das câmeras) permite estimar a profundidade dos elementos da cena, obtendo assim um mapa de profundidade (*depth map*). A câmera estéreo pode ser um sensor RGB-D (provendo cor RGB + *Depth*), ou seja, para cada *pixel* da imagem colorida adquirida (RGB) é apresentada também a sua respectiva profundidade na cena (distância para o sensor).

Outro importante sensor 3D que vem sendo muito adotado na robótica é o Kinect. Este sensor foi desenvolvido inicialmente para o *videogame* XBox 360 da Microsoft, mas posteriormente foram desenvolvidos drivers que permitem coletar seus dados por meio de diversos sistemas operacionais. Através dele são obtidos dados RGB-D da cena. O Kinect calcula a profundidade utilizando um emissor de um padrão quadriculado em NIR (*Near Infrared*) no ambiente e é capturada por um sensor NIR que processa a luz estruturada, obtendo assim a profundidade de cada *pixel* da imagem. A imagem RGB da cena é adquirida por uma câmera comum também presente no dispositivo, onde é feito o registro das imagens da câmera NIR com a câmera RGB. O Kinect, no entanto, foi desenvolvido para ser utilizado apenas em ambientes internos, onde não há a incidência direta da luz do sol, já que o sol emite luz infravermelha no mesmo comprimento de onda emitido/captado pelo seu sensor NIR, inviabilizando a obtenção da profundidade. Na Figura 6.2(h), o Kinect é mostrado.

Sensores 3D também podem ser baseados em laser multi-camadas (com múltiplos feixes), como é o caso do Velodyne. O Velodyne é um sensor que vem sendo utilizado em veículos autônomos, como o CaRINA [Fernandes et al. 2014] e o *Google Self-Driving Car*, para se detectar obstáculos no entorno do veículo, ou mesmo, para a construção de mapas. O Velodyne utiliza vários feixes lasers (podendo ser 32 ou 64 dependendo do modelo), que são rotacionados pelo dispositivo, alcançando 360 graus de campo de visão. Com base nas distâncias obtidas por cada um dos feixes é possível criar uma nuvem de pontos 3D que representa o ambiente. Os lasers do Velodyne podem medir distâncias a partir de 2 cm e podendo chegar a 120 metros em condições favoráveis [Siegwart et al. 2011]. O Velodyne HDL-32E [Velodyne LiDAR 2015] é mostrado na Figura 6.2(i).

Diversos sensores citados nesta seção estão disponíveis na sua forma simulada (bastante realista) junto ao simulador V-REP: Sonar, Lasers (Hokuyo URG-04 e Sick LMS-200, Velodyne), Câmeras, Kinect, GPS, Acelerômetro e Giroscópio. Além destes, outros sensores como os *bumbers* (sensores de contato) e de detecção de passagem também estão disponíveis no simulador.

6.2.2. Atuadores

Atuadores dotam os robôs da capacidade de produzir ações, ou seja, deslocar, manipular e interagir com o ambiente e seus elementos. Existem muitos tipos de atuadores robóticos (motores) disponíveis atualmente [Mataric 2014], dentre eles, pode-se citar: (*i*) os rotatórios - os motores do tipo servo que permitem um controle fino do posicionamento; motores de corrente contínua que são empregados no deslocamento baseados em rodas ou esteiras (avança e recua); e o motor de passo que é preciso no controle angular de giro; (*ii*) os lineares - motores que criam um movimento linear; (*iii*) os pneumáticos; e (*iv*) os hidráulicos.

Os atuadores serão dispostos de modo a garantir os movimentos das juntas dos

(a) HC-SR04 (b) URG-04LX-UG01 (c) LMS 200

(d) Point Grey Chameleon (e) Giroscópio (f) MMA7361L

(g) MPU6050 (h) Kinect (i) HDL-32E

Figura 6.2. Exemplo de sensores: (a) sonar HC-SR04 [Cytron Technologies 2015], os sensores laser (b) URG-04LX-UG01 [Hokuyo Automatic 2015] e (c) LMS 200 [SICK 2015], (d) câmera Point Grey Chameleon [Point Grey Research 2015], (e) Giroscópio mecânico, (f) acelerômetro MMA7361L [Freescale Semiconductor 2015], (g) IMU MPU6050 [InvenSense 2015], (h) Microsoft Kinect e (h) Velodyne HDL-32E [Velodyne LiDAR 2015].

robôs, sejam estes movimentos circulares contínuos (p.ex. em rodas), angulares (p.ex. em juntas de braços e pernas) ou lineares (p.ex. pistão). Os atuadores irão definir os graus de liberdade de movimento do robôs, sendo adotados de acordo com a configuração e a tarefas específicas definida para o robô, bem como possuindo uma precisão e desempenhos (velocidade, aceleração, torque, capacidade de carga) próprios.

No caso dos robôs móveis, a forma como estes atuadores são usados para controlar o seu deslocamento irá definir a sua movimentação (cinemática) e a aplicação de movimento aos motores irá resultar em uma trajetória própria de acordo com esta configu-

ração, como por exemplo, em um veículo que utiliza uma tração única na traseira (atuador das rodas traseiras) e direcionamento por uma barra de direção frontal (atuador de giro da direção, ou, de esterçamento).

6.2.3. Cinemática e Dinâmica de Robôs Móveis

A cinemática e a dinâmica podem ser estudadas separadamente, mas é interessante, quando se objetiva a análise do comportamento do robô, seu estudo em conjunto [Vieira and Roqueiro 2014].

A cinemática estuda o movimento, velocidade e acelerações de robôs móveis e articulados, assim, torna-se possível descrever a trajetória necessária para o deslocamento de um robô de uma pose (posição e orientação) inicial até a pose final. Nos robôs móveis com rodas, é usual a adoção de modelos com cinemática diferencial (controle independente de 2 rodas), permitindo que o robô gire ao redor de sua própria base (Figura 6.1(c)) e Figura 6.1(e)), por outro lado outro modelo cinemático bastante adotado é o dos veículos com barra de direção (com aceleração, freio e esterçamento), denominado de cinemática do tipo Ackermann [Dudek and Jenkin 2000]. O deslocamento de robôs com patas e pernas permite criar uma movimentação própria que se diferencia bastante da cinemática dos robôs móveis com rodas, assim como os veículos baseados em propulsão (aquáticos e aéreos).

A dinâmica também se preocupa com o movimento, velocidade e aceleração do robô, mas vai além, seu intuito está em estudar as forças e momentos envolvidos, incluindo atrito, gravidade, colisões e a reação as colisões. As equações de dinâmica do movimento são bases para inúmeros algoritmos computacionais de projeto mecânico, controle e simulação.

Tanto a cinemática como a dinâmica levam em conta as especificidades de cada robô, ou seja, a forma como se locomovem (rodas, esteiras, esferas, pernas ou pistões) e os tipos de atuadores empregados na efetivação dos movimentos. O simulador V-REP permite que sejam simulados os modelos tanto cinemático quanto dinâmico (física dos movimentos) em robôs móveis e articulados, podendo simular robôs com cinemática diferencial, Ackermann, robôs com esteiras, pernas e patas, e inclusive robôs aéreos. Uma vez que o modelo de simulação adotado respeita a cinemática e dinâmica dos robôs, isto significa que um robô pode cair, desequilibrar-se, onde podemos inclusive simular situações como a de um robô batendo em um obstáculo que pode se mover (ser empurrado) ou pode ser estático (bloco pesado de concreto).

6.2.4. Arquiteturas de Controle

Na robótica móvel, a arquitetura é a maneira pela qual se constrói um software que controla de maneira inteligente um robô [Grassi Jr. and Okamoto Jr. 2014]. A arquitetura apresenta portanto os módulos necessários para o funcionamento do sistema e de que maneira esses módulos interagem.

Há três grupos principais de módulos e componentes [Iyengar and Elfes 1991]. O grupo da Percepção envolve atividades de interpretação de sensores, modelagem do ambiente e reconhecimento. Já o grupo de Planejamento é responsável por planejar as tarefas, sincronizar e monitorar o funcionamento do robô. Por fim, o grupo da Atuação

executa os movimentos e ações propriamente ditos, controlando portanto, os atuadores.

O ciclo de tarefas composto por esses três componentes é mostrado na Figura 6.3. Na etapa de percepção o robô faz a detecção dos obstáculos por meio de seus sensores, em seguida mapeia os obstáculos criando um modelo interno para o ambiente e, por fim, atualiza sua própria posição (localização). Com base nessas informações, passa a buscar uma trajetória livre de colisão para chegar ao seu destino, na etapa de planejamento. E então, ações são tomadas por meios dos atuadores (ativação de motores), fazendo com que o robô avance, recue ou efetue curvas no momento planejado. Sendo o ambiente dinâmico, essa sequência de tarefas deve ser repetida indefinidamente.

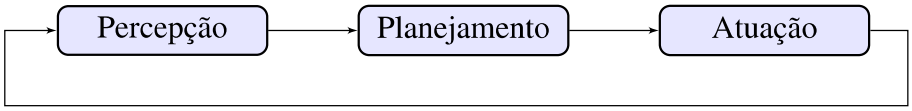


Figura 6.3. Ciclo percepção-planejamento-atuação [Grassi Jr. and Okamoto Jr. 2014].

A arquitetura de um robô pode ser classificada em: deliberativa, reativa e híbrida (combinando a deliberativa e a reativa) [Arkin and Balch 1997, Murphy 2000]. A deliberativa utiliza um modelo interno do mundo para planejar antecipadamente (deliberando) as ações do robô a fim de atingir seu objetivo. A arquitetura deliberativa se baseia no contexto e em conhecimentos sobre o mundo e a situação em que se encontra o robô. A arquitetura reativa toma ações em resposta imediata a uma percepção (reação sensorial-motora), com reações predefinidas ao se deparar com uma informação sensorial local (por exemplo, um obstáculo). Uma arquitetura híbrida faz uso do de conhecimentos de mais alto nível (p.ex. mapas e informações do nível deliberativo) para planejar suas ações, mas pode ter também a capacidade de reagir a situações inesperadas e percepções do estado atual do ambiente (p.ex. reagindo a estas percepções e elementos dinâmicos do ambiente).

Portanto, as arquiteturas reativas são fortemente indicadas a ambientes dinâmicos, as arquiteturas deliberativas são mais gerais e flexíveis na definição de ações [Arkin 1989, Mataric 1992] (precisam de pouca adaptação para funcionar em ambientes já modelados), e as arquiteturas híbridas visam extrair o melhor da deliberação e reação para a tarefa afim. Na Figura 6.4, é mostrado um diagrama entre as classificações de arquiteturas.

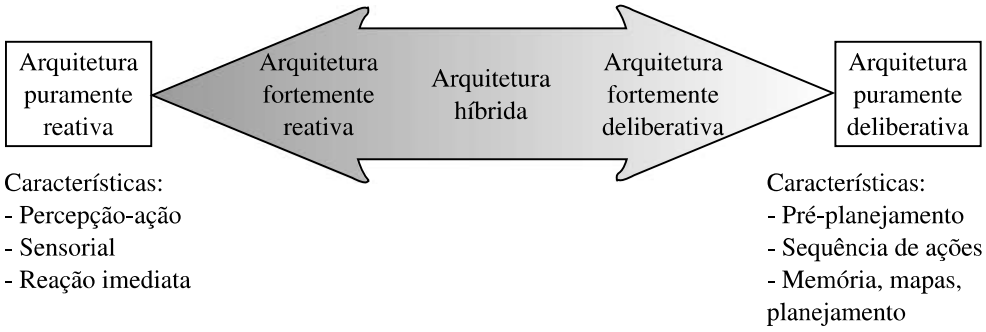


Figura 6.4. Classificação de arquiteturas baseada em deliberação e reatividade [Grassi Jr. and Okamoto Jr. 2014].

6.2.5. Comportamentos

Um robô pode ter diferentes tipos de comportamentos, onde inicialmente é necessário definir o seu grau de autonomia. Os robôs mais simples são aqueles que apenas executam de forma repetitiva ações sem considerar informações adicionais, os autômatos, seguidos pelos tele-operados, onde toda a percepção e decisão é humana e apenas a ação é executada pelo robô. Os robôs mais sofisticados, e que são o foco mais central deste texto, são os robôs que possuem um certo grau de autonomia, ou seja, possuem a capacidade de perceber o ambiente e agir de acordo com esta sua percepção e planos para executar uma determinada tarefa Figura 6.5.

	Percepção	Decisão	Ação
	Sensores	Processamento	Motores
Autômato	-	-	R
Tele-Operado	H	H	R
Semi-Autônomo	H/R	H/R	R
Autônomo	R	R	R

H: Humano – R: Robótico

Figura 6.5. Níveis de autonomia de robôs móveis e articulados

Quando se pensa em projetos de robôs autônomos, a arquitetura mais simples que pode ser adotada são os comportamentos reativos. Por meio de alguns sistemas simples e alguma preparação do ambiente é possível fazer com o robô realize tarefas importantes para empresas ou para o cotidiano das pessoas.

Uma dos comportamentos reativos muito utilizados é o do "seguidor de linha" (*Line-following behavior*). Consiste em fazer com que o robô tenha a capacidade de seguir uma simples linha no chão, e faz assim com que ele consiga na verdade seguir rotas pré-definidas, mas sem a necessidade de conhecer o mapa completo do ambiente e nem a sua localização precisa. Assim pode-se realizar uma série de operações como a de carregar mercadorias dentro de um almoxarifado ou até mesmo de um galpão de enormes proporções como o da gigante Amazon. O estoque de mercadorias da Amazon é controlado por robôs seguidores de linhas da Kiva Systems, e esta dependência destes robôs levou inclusive a gigante do varejo a adquirir a fabricante destes em 2012 [Guizzo 2012]. A Figura 6.6 mostra os robôs seguidores de linhas em operação.

O Roomba (ver Figura 6.1(e)) da iRobot [iRobot 2015] é outro exemplo de robô reativo, ele é capaz de aspirar o pó de um local utilizando rotas quase totalmente aleatórias (*Wander behavior*) apenas vagando pelo ambiente e evitando de colidir com os obstáculos por meio de seu sensoriamento, e fazendo com que busque cobrir todas as partes de uma sala pelo menos uma vez (e sem conhecer previamente o local!). O Roomba e praticamente todos os robôs baseados em comportamentos reativos possuem detecção de obstáculos, assim ele são capazes de se adaptar ao mundo dinâmico ao seu redor, sem colidir contra os elementos do ambiente (estáticos ou dinâmicos).

Figura 6.6. Robôs seguidores de linha que efetuam o transporte de mercadorias na Amazon [Guizzo 2012].

Seguir paredes (*Wall-following behavior*) também é uma estratégia reativa inteligente, já que com esta simples ideia é possível percorrer todo um andar de um edifício. Essa estratégia pode ser empregada em robôs na geração de um mapa do ambiente em que está inserido e, posteriormente, é possível navegar de maneira deliberativa usando o mapa assim construído.

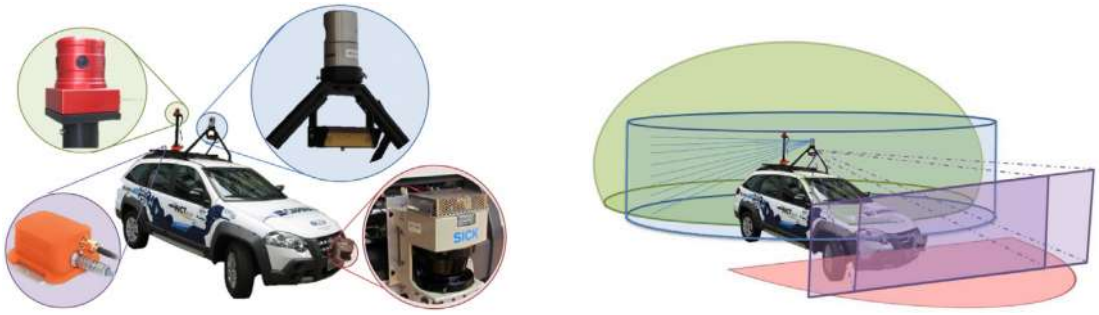
Outro comportamento interessante é o de acompanhar uma pessoa (*Follow-me behavior*) [Correa 2013]. Há alguns robôs que precisam interagir com pessoas, para esses robôs é muito importante que tenham a capacidade de acompanhar uma pessoa ao caminhar e se deslocar pelo ambiente. Um robô também pode seguir um outro robô/veículo (autônomo ou não) ou outros elementos móveis presentes no ambiente.

Inclusive podemos combinar comportamentos reativos (por fusão ou por seleção/arbitramento [Mataric 2014]) a fim de criar comportamentos mais complexos, como por exemplo, combinar um comportamento de navegação em direção a um alvo (p.ex. usando uma bússola ou uma coordenada GPS) e um comportamento de desvio de obstáculos.

Um exemplo interessante de um projeto avançado de robôs móveis inteligentes é o do projeto CaRINA [Fernandes et al. 2014], que é uma plataforma robótica utilizada no desenvolvimento de sistemas de percepção, controle e tomada de decisão para navegação autônoma e assistiva de veículos em ambientes urbanos. Ele conta com um controle computacional de esterçamento, aceleração e frenagem, e diversos sensores como: GPS, IMU, câmeras e lasers. A primeira versão do sistema autônomo, do CaRINA I (veículo elétrico), funcionava com um sistema de visão capaz de classificar imagens em "zona navegável"(asfalto) e "não navegável"(calçada, obstáculos), podendo ser orientado por coordenadas de GPS. De certo modo, podemos até dizer que este era um "seguidor de linhas"(asfalto) com direcionamento por GPS. Este sistema permitiu uma navegação por mais de 1Km no Campus da USP São Carlos usando apenas 7 pontos de coordenadas GPS como referência e se mantendo "dentro da linha da rua", tendo este experimento sido realizado em Outubro de 2011.

Certamente que para os robôs poderem desempenhar tarefas em ambientes mais

complexos e dinâmicos é interessante e importante a presença de comportamentos deliberativos e reativos (híbrido) em conjunto. Um exemplo desse tipo de projeto mais complexo é o CaRINA II (carro de passeio automatizado e autônomo) [Fernandes et al. 2014], o qual é uma plataforma robótica que possui a capacidade de navegação autônoma em ambiente urbano, tendo sido o primeiro veículo completamente autônomo a navegar em ruas urbanas, misturado ao tráfego local, no Brasil. O CaRINA 2 possui uma gama de sensores como GPS, IMU, câmeras e *lasers*, que são mostrados na Figura 6.7, que facilitam a sua percepção do mundo. Através dos dados provenientes desses sensores, alguns sistemas percebem o ambiente exterior de maneira dinâmica, como: detecção de obstáculos, detecção de pista de rodagem, detecção de cruzamentos, detecção de meio-fio, detecção de tráfego e detecção da região navegável pelo veículo. As rotas do CaRINA 2 são geradas por meio de um *waypoint* de coordenadas GPS, onde são usados mapas adaptados de sua trajetória, levando-se em conta a sua localização e as informações auferidas do ambiente pelos sistemas de percepção. O CaRINA 2 usa o sistema ROS (Robot Operating System) para a sua operação e possui uma versão de simulação realística do veículo usada para testes.



(a) Sensores (b) Percepção

Figura 6.7. Carro Robótico Inteligente para Navegação Autônoma - CaRINA 2 - visão sensorial [Fernandes et al. 2014].

6.3. Simulação de Robôs

O uso de simuladores robóticos vem crescendo muito nos últimos anos. A simulação vem sendo usada para validar e contribuir para o desenvolvimento de algoritmos robóticos. Os avanços obtido no desenvolvimento de jogos eletrônicos vem sendo incorporado à simulação nos últimos anos, permitindo um maior realismo gráfico e nos cálculos de simulação física envolvidos, resultando em uma alta qualidade de simulação da cena [Harris and Conrad 2011]. Atualmente os simuladores já tem capacidade de simular praticamente qualquer tipo de robô, sensores e ambientes. Alguns simuladores tem se destacado no cenário da robótica (Player/Stage, Gazebo, Morse, V-REP, Webots, MRS-Microsoft Robotics Simulator, Matlab Robotics Toolbox), onde as principais ferramentas livres para uso acadêmico serão listadas a seguir.

6.3.1. Player / Stage / Gazebo

O Projeto Player foi iniciado em 1999 por pesquisadores da University of Southern California, e posteriormente foram agregados pesquisadores de diversas instituições, sendo

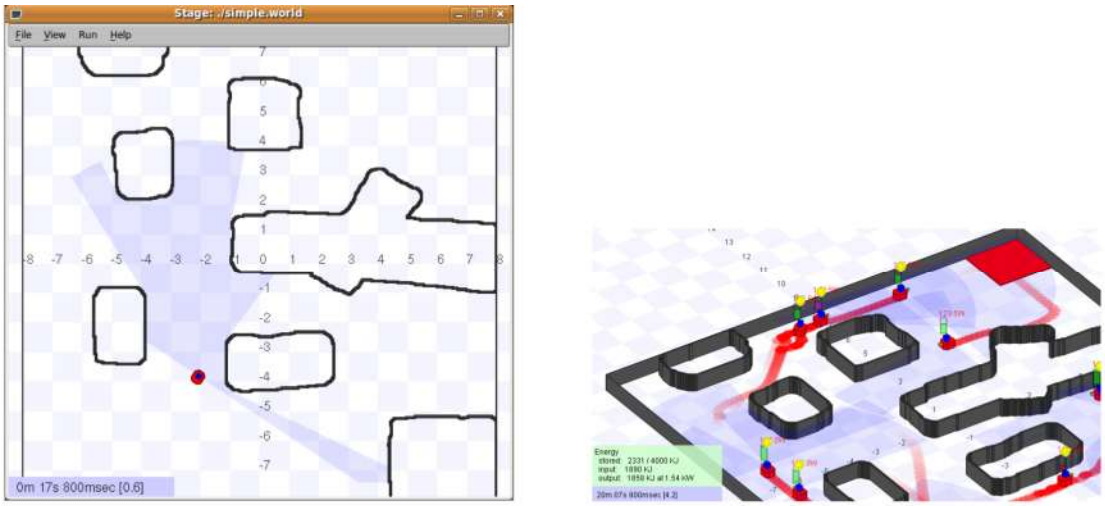
um dos ambientes mais amplamente usados em pesquisa. O Player/Stage/Gazebo tiveram uma importante contribuição junto ao ROS (Robot Operating System). O projeto é código aberto e de distribuição livre (*Open Source*), e dessa forma, está constantemente sendo desenvolvido e adequado para um número cada vez maior de robôs e sensores comercialmente acessíveis. O Player está estruturado em uma arquitetura do tipo cliente e servidor, ou seja, baseado em trocas de mensagens por uma rede de computadores (TCP/IP). A interface com o robô e seus sensores é feita por meio do servidor. O servidor obtém os dados (robô e sensores), disponibilizando-os para o cliente e, ainda, recebe instruções do cliente e as transmite ao robô. Usualmente o programa que controla o robô é o cliente (controlador "inteligente"), sendo responsável por receber os dados do servidor, processá-los e enviar as instruções para o servidor, que interfaceia e aciona o hardware do robô. O servidor funciona como uma camada de abstração, ou seja, por meio dele é possível que o mesmo cliente controle diferentes tipos de robôs, sem a necessidade de alteração de seu código, ou mesmo, que controle diversos robôs e sensores (servidores) ao mesmo tempo [Staranowicz and Mariottini 2011].

Existem muitas bibliotecas compatíveis com o cliente/servidor do Player, dentre elas pode-se citar as linguagens: C, C++, Java e Python. As bibliotecas do Player, além de possibilitar o controle do robô, possuem algoritmos de desvio de obstáculos, planejamento de trajetória e mapeamento de ambientes.

O Stage é um simulador robótico bidimensional voltado, principalmente, para ambientes internos. Permite a simulação de múltiplos robôs e sensores ao mesmo tempo, utilizando um ou mais clientes. Com o Stage pode-se simular em 2D e 1/2, ou seja, adicionar perspectiva na visualização e uma falsa impressão de 3D a simulação, porém toda simulação é feita em 2D (sensores, atuação). Na Figura 6.8, são mostrados os modos 2D e 2.5D do Stage. O Pioneer [Adept MobileRobots 2015] é o robô mais comumente adotado no Stage, inclusive por ele ser muito utilizado em laboratórios de pesquisa. A simulação do Stage abrange o deslocamento dos robôs e sensores como: odômetros, *lasers*, sonares, câmeras e garras. Os sensores no Stage se comunicam exatamente da mesma forma como seria feito com o hardware real, permitindo que o mesmo código testado em simulação seja utilizado na prática com um robô real [Vaughan et al. 2003]. Contudo, não existem garantias de que a simulação irá possuir uma fidelidade nos cálculos e comportamentos físicos [Gerkey et al. 2003].

Gazebo [4] é outro simulador compatível com o Player [Koenig and Howard 2004], porém, cria um mundo 3D, ou seja, é mais realista e pode criar ambientes mais complexos que o Stage. Todas as funções utilizadas no Player/Stage podem ser utilizadas no Player/Gazebo sem qualquer modificação. O Gazebo utiliza as bibliotecas gráficas (OGRE) e de modelagem e simulação Física (ODE) que permitem uma fidelidade do comportamento e interações físicas entre robôs e objetos do ambiente [Craighead et al. 2008]. Todos os objetos da simulação possuem massa, fricção e uma série de atributos que permitem aos objetos um maior realismo ao serem empurrados, puxados, derrubados ou transportados [Staranowicz and Mariottini 2011]. O Gazebo deve ser empregado apenas quando a simulação 2D do Stage é insuficiente para o desenvolvimento dos experimentos, já que, o Stage é menos custoso computacionalmente. Portanto, seu uso torna-se inevi-

[4]Gazebo Simulator: http://gazebosim.org/



(a) 2D (b) 2,5D

Figura 6.8. Exemplo de simulação Player/Stage, onde, (a) mostra o Stage em 2D e (b) o Stage em 2.5D. [Fonte: http://playerstage.sourceforge.net/?src=stage]

tável para ambientes externos (com solo irregular) ou quando a altura dos objetos ou a posição e orientação dos sensores deve ser levada em consideração na simulação do robô. A Figura 6.9 mostra um exemplo da simulação Player/Gazebo.

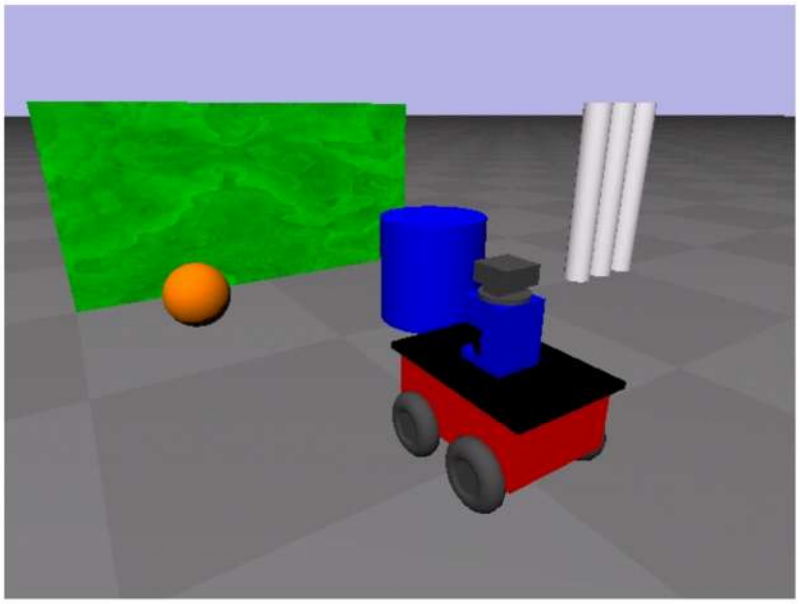


Figura 6.9. Exemplo de simulação Player/Gazebo. [Fonte: http://playerstage.sourceforge.net/gazebo/gazebo.html]

6.3.2. ROS

ROS (Robot Operating System) [5] é atualmente uma das mais populares ferramentas da robótica, possuindo licença BSD (código aberto) [Quigley et al. 2009]. Seu suporte

[5]ROS - Robot Operating System: http://www.ros.org/

abrange diversos tipos de robôs e sensores, funcionando com baseado em serviços (*publish/subscribe*). O ROS possui uma comunidade muito ativa, disponibilizando uma série de pacotes e códigos prontos, o que facilita o seu desenvolvimento. O ROs é basicamente um "sistema operacional robótico"feito para permitir o acesso as diversos equipamentos e dispositivos robóticos (sensores, atuadores) e provendo também inúmeras bibliotecas de módulos avançados para o controle dos robôs. Portanto, o ROS é usado principalmente com robôs reais.

Por outro lado, para a simulação é possível usar o ROS em conjunto com alguns simuladores como, por exemplo, o Gazebo (ver Seção 6.3.1), incluindo as funções de mais alto nível disponíveis para o próprio ROS e Player. Deste modo, o ROS passa a controlar os robôs, sensores e atuadores simulados pelo Gazebo.

6.3.3. MORSE / Blender

MORSE (*Modular OpenRobots Simulation Engine* [6]) [Echeverria et al. 2011] é uma proposta acadêmica de simulador robótico, com desenvolvimento iniciado pelo *Laboratoire d'Analyse et d'Architecture des Systèmes* da Universidade de Toulouse, França. ROS (ver Seção 6.3.2, YARP (Yet Another Robot Platform) [Metta et al. 2006], Pocolibs [Pocolibs 2015], e MOOS [Oxford Mobile Robotics 2015] são plataformas robóticas que podem ser utilizadas em conjunto com a simulação do MORSE, permitindo assim o controle de robôs móveis em ambientes Terrestres, Aquáticos e Aéreos. É possível utilizar uma gama de sensores na simulação, dentre eles: acelerômetros, *lasers*, câmeras, e câmeras que capturem a profundidade. Com o MORSE, pode-se ainda criar facilmente componentes customizados como por exemplo, sensores e atuadores, que não acompanhem o pacote de instalação convencional. Toda a programação e customizações do MORSE podem ser feitas por meio de *scripts* em Python. As visualizações e simulações dos ambientes 3D são baseadas no Blender (OpenGL) e a fidelidade física é garantida pelo emprego do motor Bullet [Cook et al. 2014] (engine física). Um exemplo de simulação no MORSE é mostrada na Figura 6.10.

Figura 6.10. Exemplo de simulação no MORSE. [Fonte: https://www.youtube.com/watch?v=kGx5SzQ3YWQ]

[6]MORSE: http://www.openrobots.org/morse/

6.3.4. V-Rep

O V-REP [7] é um simulador robótico 3D desenvolvido pela Coppelia Robotics. O simulador possui dupla licença, para utilização comercial é necessária à aquisição de licença e para uso educacional é gratuito (Free Educational e GNU GPL [8]). Ele suporta múltiplos motores de simulação física (Engines ODE, Bullet, Vortex), permitindo inclusive a troca durante a simulação (em tempo real). Além de prover a simulação, o V-REP [E. Rohmer 2013] permite a integração do desenvolvimento dos controladores do robô e customização do ambiente utilizando a linguagem de script Lua. É possível ainda programar a simulação usando *plugins*, ROS e com APIs disponíveis em diversas linguagens de programação como C/C++, Java, Python e Matlab.

O V-REP possui suporte a diversos tipos de robôs móveis e articulados (p.ex.: Pioneer, E-Puck, Khepera, Kuka Youbot, NAO, Baxter e manipuladores ABB, Kuka e Adept), atuadores e diversos sensores (p.ex.: *lasers*, câmeras, Kinect, Velodyne, GPS, acelerômetros) sendo possível ainda o desenvolvimento de novos atuadores e sensores via *plugins*. O simulador suporta robôs terrestres, aquáticos e inclusive aéreos, porém, para utilização de robôs aquáticos é necessária à criação de um ambiente apropriado. Na Seção 6.4 são apresentados mais detalhes do V-REP, que foi o simulador escolhido para ser adotado neste curso. As principais vantagens do V-REP são: a sua simplicidade de uso, a flexibilidade de programação e interfaces, a enorme variedade de modelos de robôs e equipamentos simulados, a facilidade de edição das cenas para a simulação, a disponibilidade para diferentes plataformas de Hardware (Windows, Linux e Mac - sem requisitos especiais em termos de configuração de processador, memória e/ou placa gráfica), e por ser distribuído de forma gratuíta e aberta ao desenvolvimento para o uso acadêmico/educacional.

6.4. Uso e Desenvolvimento de Aplicações no Simulador V-REP

O Simulador V-REP permite simular diferentes tipos de bases robóticas, que incluem robôs como o Pioneer P3DX [Adept MobileRobots 2015] com cinemática diferencial, robôs de cinemática Ackermann (tipo carro com barra de direção, aceleração e freio), robôs humanoides como o NAO [Aldebaran Robotics 2015], robôs articulados com patas, braços robóticos manipuladores entre outros modelos disponíveis. O V-REP possui ainda uma ampla gama de sensores *laser* (Hokuyo,Sick,Velodyne), câmeras, Kinect, GPS, etc. A Figura 6.11 mostra um exemplo de tela do simulador[9], incluindo 3 robôs: NAO, Pioneer, Spider e um Kinect sobre uma mesa. Vários exemplos de cenas e simulações usando o V-REP relacionados a este curso podem ser encontrados na Internet [10].

É possível com o V-REP modelar o ambiente e os robôs, onde podemos adicionar objetos, paredes e outros elementos no ambiente virtual 3D, assim como podemos também criar novos robôs adicionando componentes, como por exemplo, é possível adicionar facilmente um sensor *laser* Sick ou Hokuyo ao robô Pioneer apresentado na Figura 6.11. Na Seção 6.4.1 é apresentada a interface do V-REP com o usuário, mostrando suas prin-

[7] V-Rep: http://www.coppeliarobotics.com/

[8] V-Rep Licensing: http://www.coppeliarobotics.com/licensing.html

[9] Vídeo da simulação desta cena está disponível em `https://youtu.be/V_4vFyGGEDg`.

[10] V-REP Curso JAI: https://www.sites.google.com/site/vrepjai/

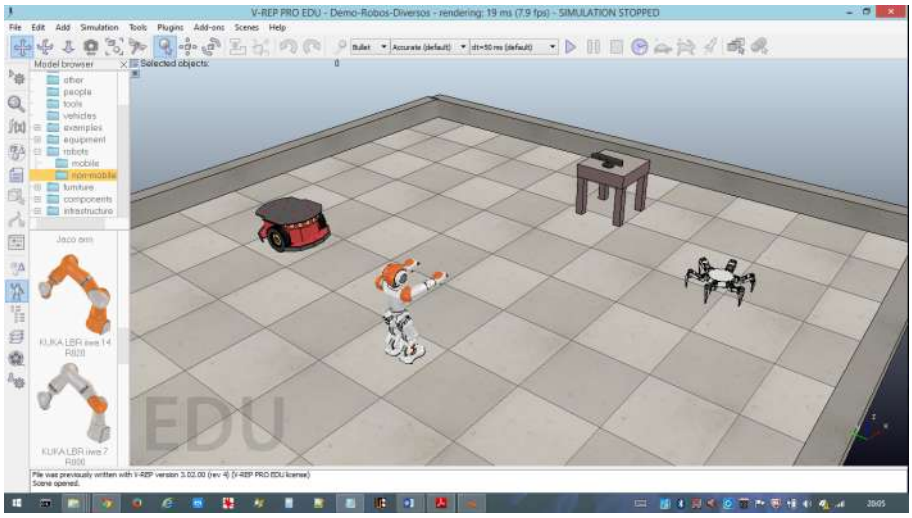


Figura 6.11. Simulador V-REP com ambiente 3D e Robôs Móveis/Articulados.

cipais funções e ferramentas. Particularidades sobre a cena e modelos empregados para a simulação são explanados na Seção 6.4.2. A Seção 6.4.3 apresenta algumas informações adicionais sobre o controle da simulação e quais são os parâmetros principais envolvidos. Além disto, é apresentada na Seção 6.4.4 os tipos e meios de se programar no V-REP, ou seja, como desenvolver programas que possam acessar os dados dos sensores, processar estes dados e enviar comandos para os motores do robô.

6.4.1. Interface com o usuário

A aplicação V-REP é composta por uma janela de console e uma janela de aplicação. Por meio da janela de console é executado o V-REP e são mostradas as impressões em tela do Lua, quais plugins estão sendo carregados e se foram corretamente inicializados, ou seja, é exibido todo o registro de informação relevante do sistema. Somente no Linux é obrigatória a execução do V-REP partindo-se de um terminal, porém, é uma prática recomendável em certas circunstâncias para os outros sistemas operacionais também.

A janela de aplicação é onde a cena é mostrada, editada, simulada e onde são feitas as interações com o sistema e a simulação. O usuário pode abrir muitas cenas em paralelo, mas somente uma pode estar ativa (em simulação ou edição). A Figura 6.12 mostra um exemplo da janela de aplicação e seus componentes que são descritos a seguir:

- *Componentes Hierárquicos na Cena*: é o componente que mostra a estrutura hierárquica de todos os objetos que compõem a cena. Cada objeto possui um ícone que identifica seu tipo, um nome e, em alguns casos, há um botão para a edição do *script* Lua que o rege e para a janela de parâmetros do *script* (Fig. 6.13). Clicando uma vez sobre o objeto ele é selecionado e destacado na cena. Ao clicar duas vezes sobre o ícone de um objeto a janela de propriedades do objeto é aberta. A habilitação da edição do nome do objeto se dá ao clicar duas vezes sobre seu nome. Objetos podem ser arrastados para dentro e fora de outros objetos, alterando assim o relacionamento entre os componentes da cena. O componente de hierarquia na cena pode não estar visível, já que há um botão na Barra de Ferramentas 1 permite

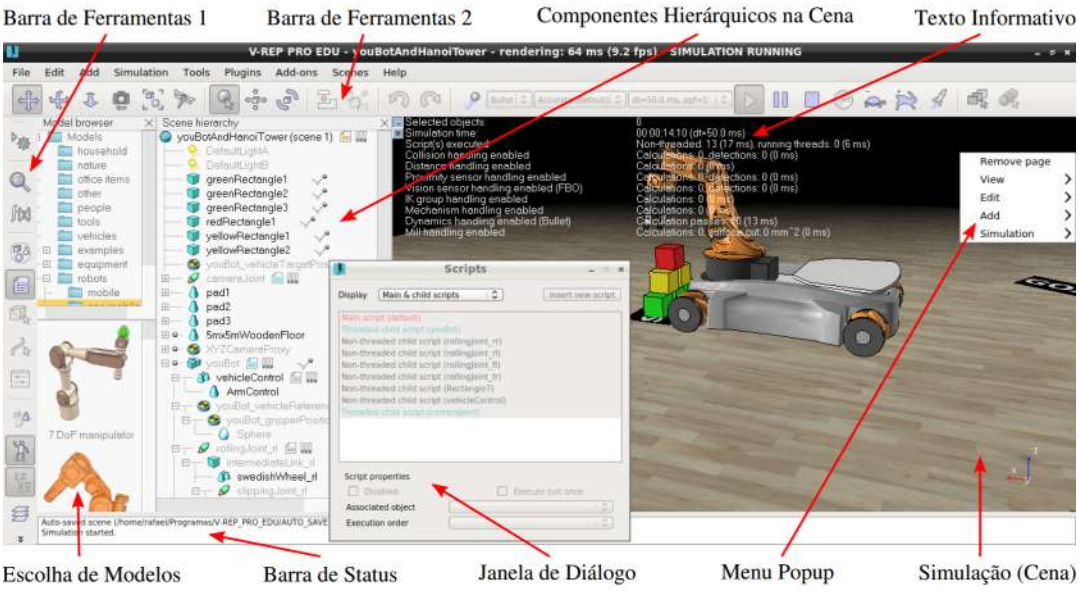


Figura 6.12. Componentes de tela do V-rep.

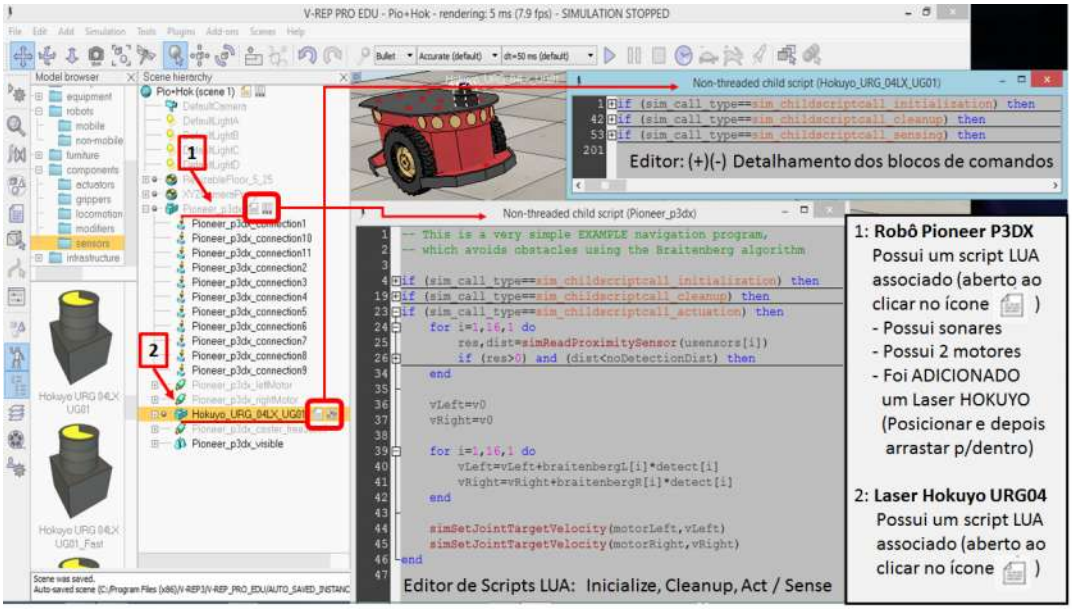


Figura 6.13. Robô e Sensores com Scripts LUA do V-rep.

que seja exibido e escondido.

- *Texto Informativo*: exibe as informações referentes ao estado e parâmetros atuais do objeto/item selecionando na cena simulada.

- *Menu Popup*: é exibido ao clicar com o botão direito do mouse. Seu conteúdo depende do componente ao qual foi clicado e funciona como atalho rápido para funções dos objetos e componentes.

- *Janela de Diálogo*: Existem ainda várias janelas auxiliares, aos quais, permitem que

o usuário edite e interaja com a cena, possibilitando a alterações das configurações e parâmetros específicos de cada objeto ou item envolvido.

- *Barra de Status*: exibe informação relativa as operações e comandos, mostrando sempre as mensagens de erros provenientes do interpretador Lua. Pode-se anda enviar alguma informação para a Barra de Status usando a função simAddStatusbarMessage.

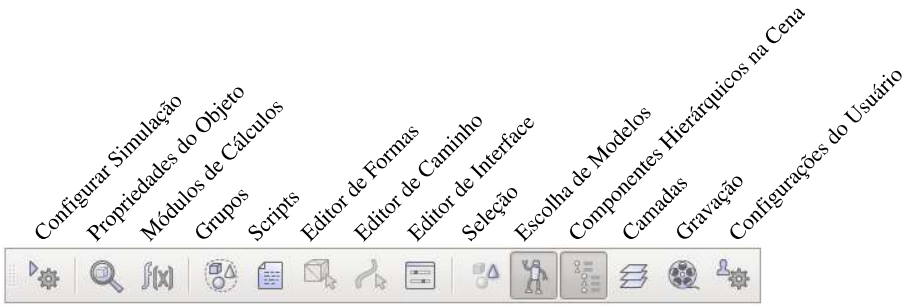
- *Escolha de Modelos*: exibe a estrutura de modelos disponíveis na instalação do V-REP. A estrutura é exibida através de uma árvore, ao qual, organiza os modelos conforme sua finalidade. Abaixo da árvore são mostradas as miniaturas de modelos pertencentes à pasta selecionada. Para incluir um modelo na cena, basta clicar sobre o modelo desejado e arrastar até a posição da cena desejada. Alguns modelos já trazem um *script* Lua padrão ao serem incluídos (Figura 6.13). O componente de escolha de modelos está visível por padrão, porém pode ser mostrado e escondido através de um botão na Barra de Ferramentas 1.

- *Barra de Ferramentas 1*: possui botões de atalho para a configuração da simulação, propriedades dos objetos, módulos de cálculo, os *Scripts* presentes na cena, edição de formas/caminho/interface, habilitação de seleção de objetos na cena, exibe/esconde o componente de Componentes Hierárquicos na Cena e Escolha de Modelos, Camadas, geração de vídeos da simulação e, por fim, as configurações globais do V-REP. A Figura 6.14(a) exibe os botões disponíveis na Barra de Ferramentas 1.

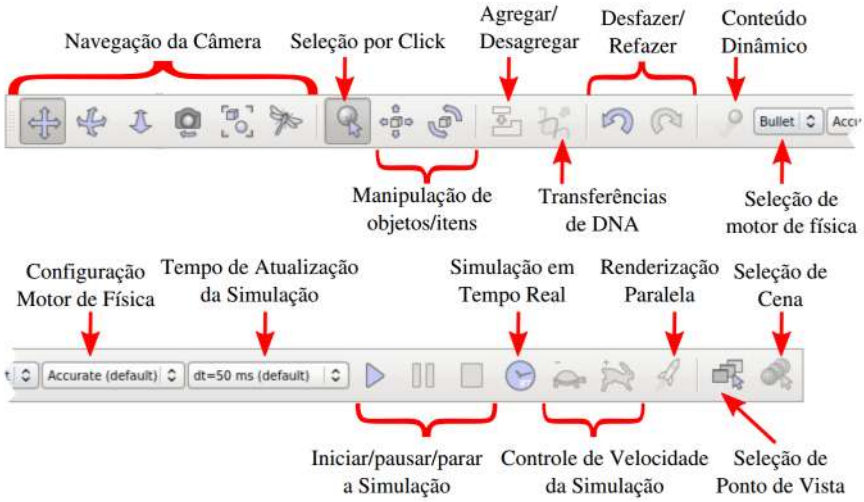
- *Barra de Ferramentas 2*: possui funções referentes ao controle da navegação da câmera (ponto de vista - camera pan, rotate e shift), habilita ou desabilita a seleção de objetos por clique, agrega ou desagrega objetos selecionados, desfazer (undo) e refazer (redo) as alterações feitas, habilitar e desabilitar o uso de modelos dinâmicos na simulação (renderização e formas mais complexas), permite o deslocamento e rotação de objetos (object shift e rotate), escolha do motor responsável pela física (inclusive durante a simulação), escolha do tempo para atualização da simulação, iniciar/pausar/parar a simulação, habilitar a simulação em tempo real (procura manter a simulação sincronizada ao tempo real discorrido), alterar a velocidade em que a simulação ocorre (habilitada somente quando a simulação não for em tempo real), permitir que o V-REP realize a renderização em paralelo, seleção de ponto de vista e seleção da cena. Na Figura 6.14(b) é mostrada a Barra de Ferramentas 2.

6.4.2. Cenas e modelos

Os principais elementos das simulações são a cena e os modelos. Um modelo nada mais é do que um subelemento da cena, sendo possível incluir em uma cena vários modelos. Somente a cena pode ser simulada. Cenas são salvas com a extensão de arquivo “ttt”, onde este arquivo contém toda a informação necessária para a reabertura e simulação da cena. Já os modelos possuem a extensão “ttm” e possuem todas as informações necessária para a criação de um objeto de seu tipo. Um modelo pode ser criado a partir da combinação de outros modelos, podendo inclusive um objeto ser importados a partir de softwares



(a) Barra de ferramentas 1



(b) Barra de ferramentas 2

Figura 6.14. Barras de ferramentas

externos de modelagem 3D. A Figura 6.15 traz um exemplo de modelo de robô Pioneer P3DX inserido na cena, ao qual, a entidade criada na cena possui propriedades próprias, customizáveis e sem vinculação com o modelo gerador.

Os elementos presentes na cena e no modelo são similares, no entanto a cena possui alguns elementos a mais que são: o ambiente, um *script* principal e os pontos de vista da cena. Nas subseções a seguir são expostas informações adicionais sobre entidades e o ambiente no V-REP.

6.4.2.1. Ambiente

O ambiente no V-REP é definido por algumas propriedades e parâmetros que fazem parte da cena, mas não são objetos da cena. Alguns parâmetros que podem ser definidos no ambiente são: cores de fundo, parâmetros para inclusão de nevoeiro, luz ambiente, informação para a criação da cena, etc. As propriedades do ambiente podem ser mostradas, ocultadas ou alteradas através do menu superior (File, Edit, Add...) na opção Tools e selecionando a seguir a opção Environment no V-REP. A tela de alteração da cena é mostrada na Figura 6.16, onde:

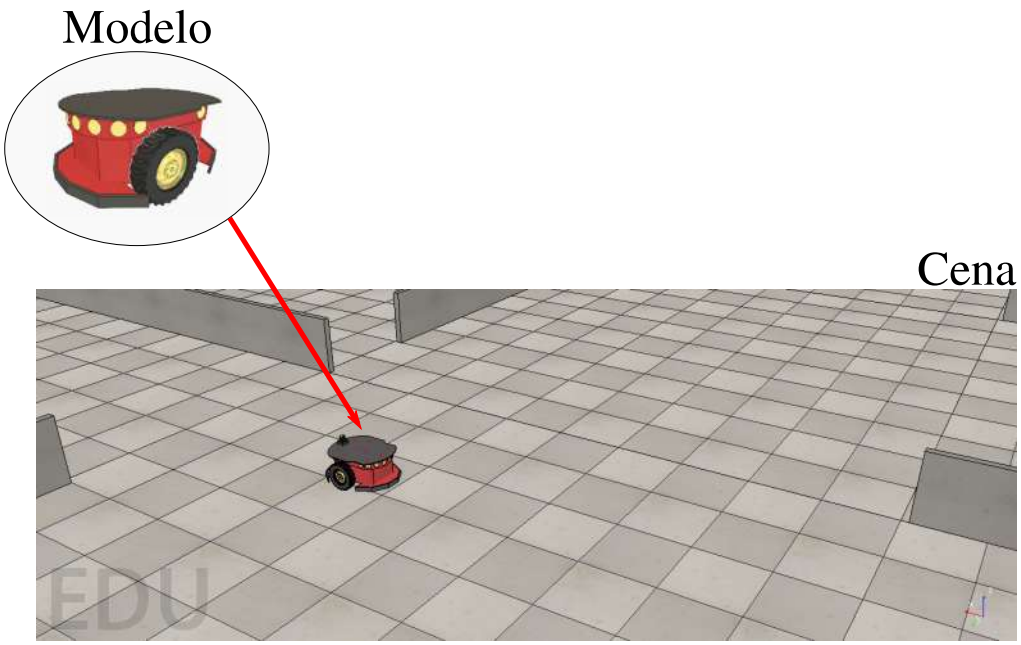


Figura 6.15. Modelo do robô Pioneer p3dx incluído na cena.

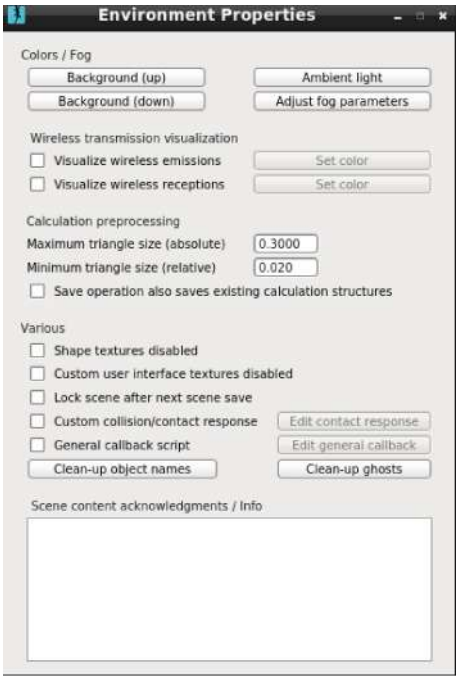


Figura 6.16. Propriedades do ambiente no V-REP.

- *Background (up / down)*: permite ajustar as cores de background da cena. Possui duas partes, a superior (céu) e os componentes correspondentes ao solo do ambiente.

- *Ambient light*: ajusta a luz da cena.

- *Adjust fog parameters*: permite adicionar nevoeiro no ambiente e assim pode-se dificultar a leitura e percepção do ambiente pelos sensores.

- *Visualize wireless emissions* e *Visualize wireless receptions*: habilita ou desabilita a visualização das comunicações sem fio.

- *Maximum triangle size* e *Minimum triangle size*: regem as dimensões aceitas para os triângulos empregados nos módulos de cálculos da simulação, mas que em nada altera a aparência dos objetos da cena. O tamanho máximo é absoluto, mas o tamanho mínimo do triângulo é relativo ao valor máximo aceito. Estes parâmetros afetam, por exemplo, o módulo de cálculo da distância mínima entre duas entidades, ou seja, quanto mais triângulos forem gerados para representar os objetos da cena maior será o custo computacional da cena.

- *Save operation also saves existing calculation structures*: opção que permite ao usuário do V-REP salvar, juntamente com a cena, as informações da estrutura de dados usada nos cálculos de colisão, distância, etc, assim é evitado algum tempo no pré-processamento da simulação. O efeito colateral é um arquivo de cena com mais informações e maior tamanho.

- *Shape textures disabled* e *Custom user interface textures disabled*: habilitam/desabilitam as texturas para formas e interfaces customizadas.

- *Lock scene after next scene save*: bloqueia a cena para edição, exportação de relatórios e visualização de *script*, sendo esse processo irreversível após o salvamento do arquivo.

- *Custom collision/contact response*: habilita/desabilita a customização do contato dinâmico da cena via script.

- *General callback script*: habilita/desabilita o retorno geral via *script*, ou seja, as chamadas feitas por um *plugin* da função da API *simHandleGeneralCallbackScript*.

- *Clean-up object names*: coloca um pouco de ordem nos nomes dos objetos.

- *Clean-up ghosts*: remove todos os objetos "fantasmas" que possam vir a estar ocultos na cena.

- *Scene content acknowledgements / Info*: permite colocar qualquer informação adicional textual que será exibida em toda abertura da cena.

6.4.2.2. Entidades

Entidade é o termo usado para um objeto ou uma coleção de objetos no V-REP.

Os objetos são os elementos usados para construir a cena da simulação, ou mesmo, na criação de modelos empregados nas cenas. Existem alguns tipos de objetos e eles podem ser facilmente reconhecidos pelo ícone presente ao lado de seu nome no componente que exibe a estrutura hierárquica dos objetos na cena (ver Componentes Hierárquicos na

Cena na Seção 6.4.1). A Figura 6.17 mostra os tipos de objetos presentes no V-REP e seus respectivos ícones.

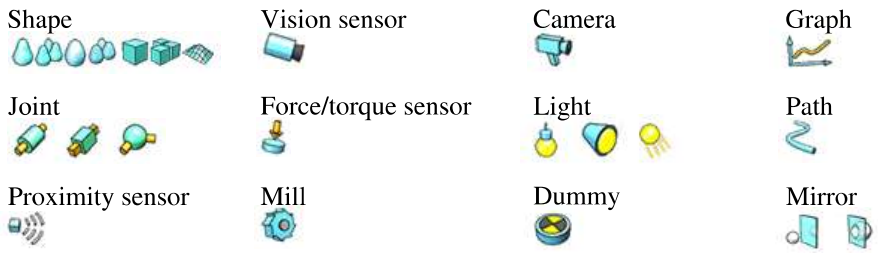


Figura 6.17. Tipos de objetos do V-REP [Coppelia Robotics 2015].

Os tipos de objetos presentes no V-REP são os seguintes:

- *Shape*: é uma forma rígida em malha composta por faces triangulares.
- *Joint*: é um atuador (ver Seção 6.2.2).
- *Graph*: é um gráfico de geração e visualização de dados da simulação.
- *Dummy*: é um ponto de orientação na cena.
- *Proximity sensor*: é um sensor que detecta a forma geométrica e volume de objetos da cena.
- *Vision sensor*: é uma câmera, ou seja, reage a luz, cores e imagens.
- *Force sensor*: sensor que mede forças e torques aplicados a ele.
- *Mill*: objeto capaz de efetuar operações de corte nos objetos shapes presentes na cena.
- *Camera*: objeto que permite a visualização da simulação por vários pontos de vistas.
- *Light*: ilumina a cena.
- *Path*: é um objeto do tipo caminho e que define uma trajetória.
- *Mirror*: é um objeto auxiliar que pode refletir imagens ou iluminação.

Alguns dos tipos de objetos possuem ainda propriedades especiais[11], as quais são levadas em conta pelos módulos de cálculos internos do V-REP, ou mesmo, regem algumas interações com outros objetos. As propriedades especiais são: *collidable* (objeto testado para colisão entre objetos), *measurable* (mensurável quanto a distância entre objetos), *detectable* (detectável por sensores de proximidade), *cuttable* (objeto cortável),

[11] As propriedades especiais (*object special properties*) são acessíveis via menu do V-REP em *Tools* / *Scene object properties* e na janela aberta escolha a opção *Common*.

renderable (pode ser visto e detectado por sensores de visão) e *viewable* (compete a objetos que são transparentes ou que podem mostrar alguma imagem).

Uma coleção de objetos (*collections*) contém um grupo de objetos que são vistos pelo V-REP como uma única entidade. Ela serve de suporte para os módulos de cálculos onde só é levada em conta uma única entidade, portanto, possibilitando a um robô evitar o teste de colisão contra vários objetos (ao invés de um apenas). A coleção de objetos possui algumas das propriedades especiais disponíveis para objetos como: *collidable*, *measurable*, *detectable*, *cuttable* e *renderable*. Porém, as propriedades da coleção não são aplicadas a todos os objetos pertencentes a coleção, ou seja, caso uma coleção seja *collidable*, somente os objetos da coleção marcados também como *collidable* serão considerados para o cálculo da colisão.

6.4.3. Simulação

Alguns botões da Barra de Ferramentas 2 (ver Seção 6.4.1) auxiliam no controle da simulação, desempenhando papéis, como por exemplo, de iniciar, pausar ou parar a simulação. É possível ainda modificar algumas configurações na tela de configurações da simulação[12] no V-REP, que é mostrada na Figura 6.18, onde:

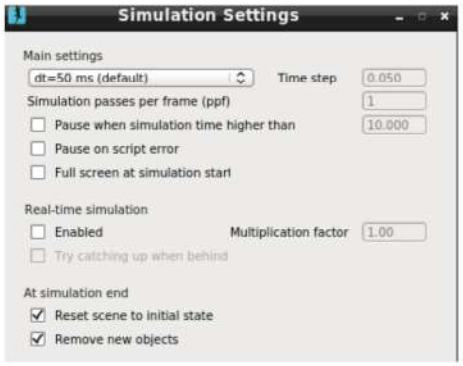


Figura 6.18. Configurações da simulação no V-REP.

- *Time step*: é o tempo decorrido da simulação em cada laço do *script* principal. Usando um tempo elevado a simulação torna-se mais rápida, porém, também apresenta-se mais instável e imprecisa. O inverso ocorre com tempos menores, a simulação torna-se mais precisa, porém mais lenta.

- *Simulation passes per frame (ppf)*: número de ciclos do *script* principal (simulação) para que seja refeita a tela da simulação, sendo comumente utilizado o valor 1 (a cada ciclo a tela é atualizada). Esta opção pode vir a ser interessante para placas gráficas com dificuldades de efetuar a visualização da simulação a contento.

- *Pause when simulation time higher than*: permite especificar em que momento a simulação será automaticamente pausada.

[12] A tela de configuração da simulação está acessível via menu do V-REP em Simulation / Simulation Settings.

- *Pause on script error*: quando habilitada a simulação é pausada automaticamente ao ocorrer erros no *script*.
- *Full screen at simulation start*: permite a execução da simulação em modo tela cheia. Durante a simulação tela cheia é possível voltar ao modo normal de exibição utilizando a tecla ESC do teclado ou via *script* alterando a variável booleana *sim_booparam_fullscreen* para falso.
- *Real-time simulation, multiplication factor*: se habilitada o V-REP tentará cumprir a execução da simulação em tempo real. E o *multiplication factor* dita quantas vezes mais rápido que o tempo real será o objetivo que o V-REP deverá perseguir na simulação.
- *Reset scene to initial state*: se selecionado, toda vez que a simulação é parada, todos os objetos voltam para o seu estado inicial, incluindo sua a posição, orientação, posição dos atuadores, etc.
- *Remove new objects*: quando selecionado, qualquer objeto adicionando durante a simulação será automaticamente removido da cena ao final dela.

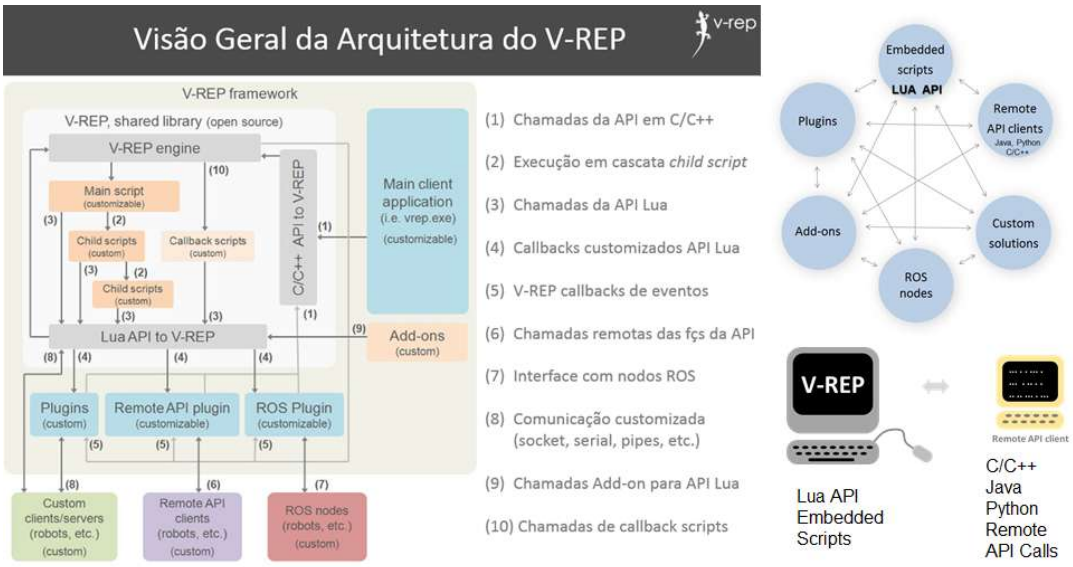


Figura 6.19. Arquitetura de Interfaces de Programação do V-REP. [Fonte: Adaptado de [E. Rohmer 2013] e V-REP Overview Presentation (Coppeliarobotics)]

6.4.4. V-REP e Programação

O V-REP é um simulador flexível e customizável, permitindo a programação de particularidades da simulação e o controle dos robôs (sensores e atuadores) utilizando, praticamente, qualquer linguagem de programação. A linguagem Lua [PUC-Rio 2015] (ver Seção 6.4.4.1) é a linguagem de programação nativa do V-REP, já estando incorporada ao simulador, permitindo a customização de toda a cena e manipulação de objetos através de seus scripts em LUA (*LUA API - Embedded Scripts*). Portanto, utilizando essa linguagem de programação se garante a compatibilidade de versões e a portabilidade de

sistema operacional da cena em V-REP, já que os *scripts* Lua são gravados juntamente com o salvamento da cena. Por outro lado, o V-REP oferece uma sofisticada arquitetura para o desenvolvimento de programas, através de diferentes interfaces, como apresentado na Figura 6.19. Portanto, existem outras formas de acesso à simulação via V-REP além dos scripts LUA, que são:

- *Add-on*: este método também utiliza Lua, porém utiliza um script em um arquivo externo à cena. Pode ser útil para casos em que se queira customizar rapidamente o simulador, pois, trocando o arquivo *script* Lua já se pode mudar o comportamento de um robô, por exemplo. O *script* Lua nesse método é iniciado automaticamente com a simulação.

- *Plugin*: pode-se desenvolver plugins para V-REP utilizando as linguagens C/C++. Um plugin é uma biblioteca que é automaticamente incorporada ao V-REP. Um *plugin* pode funcionar como extensão da linguagem lua, permitindo assim a criação de novos comandos e facilitadores para seus *scripts*. Em outros casos, pode ser usado para ganhar desempenho já que este método é o mais eficiente em termos computacionais. Pode-se desenvolver ainda um *plugin* para controlar toda a cena, como é feito com o Lua.

- *API remota*: um cliente remoto da API pode ser desenvolvido em C/C++, em Python, Java, Matlab/Octave, em um script Urbi, ou mesmo através de um *script* Lua. A API remota disponibiliza cerca de 100 funções que permitem o controle da cena e dos objetos presentes nela, utilizando para isso comunicação TCP/IP e chamadas de funções equivalentes/similares a da API Lua embutida no V-REP. O programa controlador da cena não será automaticamente inicializado, como nos casos anteriores, ou seja, é necessária a inicialização da cena no V-REP e então a execução do programa controlador da cena. Esta opção permite que o programa controlador esteja locado em qualquer computador com acesso a internet.

- *ROS node*: este método permite que uma aplicação externa conecte o V-REP usando o ROS (*Robot Operating System*), suportando assim, qualquer linguagem de programação que utilizar o ROS. A aplicação ROS não precisa necessariamente estar na mesma máquina que a simulação.

- *Cliente/servidor customizado*: pode-se criar ainda um método próprio de interação com o V-REP, mas neste caso ou o cliente ou o servidor precisam se comunicar com o V-REP utilizando um *script*, *plugin* ou utilizando algum meio de comunicação (por exemplo, *sockets*). Este é o meio mais flexível por permitir ao V-REP interagir com qualquer linguagem de programação. Porém é o método mais trabalhoso já que toda a estrutura de comunicação precisa ser recriada ou adaptada. Esta abordagem permite a criação se uma arquitetura cliente-servidor entre o V-REP e um módulo externo de controle dos robôs.

Mesmo utilizando uma linguagem diferente do Lua para a customização do V-REP, é conveniente ao usar o V-REP ter noções básicas dessa linguagem de programação,

uma vez que o LUA é a linguagem nativa de desenvolvimento junto ao V-REP. Um exemplo dessa utilidade vem da necessidade de se editar os scripts padrão de certas cenas visando eliminar comportamentos padrões pré-existentes em alguns modelos do V-REP. Portanto, caso se deseje controlar um objeto proveniente deste modelo fora do V-REP, será necessário antes apagar partes do código Lua padrão.

Nas subseções a seguir são mostradas algumas informações adicionais da linguagem Lua e de seu uso junto ao V-REP, e em seguida são apresentadas informações adicionais sobre a programação do V-REP utilizando a API remota.

6.4.4.1. Lua

Lua [PUC-Rio 2015] é uma linguagem *script* e procedural, que foi criada com o intuito de ser um facilitador para expansão de aplicações. Lua é uma linguagem case sensitive, ou seja, faz diferença entre letras maiúsculas e minúsculas. Sua criação se deu por uma equipe de desenvolvedores da Tecgraf da PUC-Rio.

As variáveis em Lua são dinamicamente tipadas, ou seja, o dado contido nela determina seu tipo. Os tipos básico de dados que uma variável pode conter são: *nil* (valor nulo e é atribuído automaticamente na criação da variável), *boolean* (*true/false*), *number* (números reais), *string* (conjunto de caracteres de 8 bits), *function* (apontamento para funções em Lua), *userdata* (bloco de memória que armazena qualquer tipo de dado), *thread* (variável que representa fluxos de execução independentes de rotinas Lua) e *table* (são arrays que podem conter dados de todos os tipos exceto nil). As variáveis podem ser globais, locais ou campos de tabelas, sendo que o Lua utiliza por padrão uma variável como global. Em Lua pode-se efetuar múltiplas atribuições de variáveis, um exemplo dessa utilização é mostrada na Figura 6.20.

```
x,y,z = myTable[1],myTable[2],myTable[3]
```

Figura 6.20. Múltiplas atribuições em Lua.

Os operadores relacionais em Lua são: == (igualdade), ∼= (negação de igualdade), < (menor que), > (maior que), <= (menor ou igual que) e >= (maior ou igual que). Na Figura 6.21 é mostrado um exemplo de controle condicional (*if*) em Lua usando o operador de igualdade. Pode-se encadear cláusulas usando as palavras-chaves *and* (E) ou *or* (OU). Usando -- pode-se declarar todo o texto até o final dessa linha como comentário, ou mesmo, empregar *--[[* para abrir e *--]]* para fechar o texto de comentário a ser ignorado pelo interpretador Lua. Os laços que podem ser utilizados em Lua são *for*, *while* e *repeat*. Na Figura 6.22 são mostrados exemplos dos laços onde são exibidos no terminal os números de 1 a 4. Nota: no V-REP devemos considerar em qual console/janela serão exibidos os dados, e muitas vezes são usadas funções da API do V-REP para esta finalidade.

Um exemplo de *script* em Lua do V-REP é mostrado na Figura 6.23, onde, é efetuado o controle de um robô Pioneer baseado no sensor Sonar/Ultrassom (*usensor*) para gerar comandos de desvio de obstáculos usando cinemática diferencial (controle indepen-

```
if value1==value2 then
  print('value1 and value2 are the same!')
end
```

Figura 6.21. Controle condicional em Lua

```
--Contando de 1 a 4 usando for.
for i=1,4,1 do
  print(i)
end

--Contando de 1 a 4 usando while.
i=0
while i~=4 do
  i=i+1
  print(i)
end

--Contando de 1 a 4 usando repeat.
i=0
repeat
  i=i+1
  print(i)
until i==4
```

Figura 6.22. Opções de Laço em Lua

dente de velocidade dos motores *vLeft* e *vRight*). Este é o código do script Lua padrão associado ao robô Pioneer P3DX disponível junto aos exemplos de modelos de robôs do V-REP. O script apresentado na Figura 6.23, como é usual nos scripts dos objetos simulados, é composto por 3 partes: Inicialização (*sim_childscriptcall_initialization*), Finalização (*sim_childscriptcall_cleanup*) e Execução (*sim_childscriptcall_actuation* no caso dos atuadores, podendo ter também um *sim_childscriptcall_sensing* no caso dos sensores). Também é importante destacar que alguns scripts possuem parâmetros que definem as propriedades dos elementos, como por exemplo, nos sensores laser Sick ou Hokuyo onde a abertura de varredura é definida através destes parâmetros (ajustados clicando no ícone ao lado direito do ícone do script Lua do objeto).

Apesar do Lua ser uma linguagem procedural, o V-REP faz chamadas nos scripts de inicialização, finalização e execução, utilizando tipos de chamadas (*sim_call_type*), ou seja, o V-REP utiliza uma programação por eventos, onde, controles condicionais encapsulam os comandos específicos dos eventos da simulação. No caso de uma simulação que inclua diferentes objetos, como por exemplo, um robô Pioneer P3DX dotado com um sensor laser Hokuyo URG-04 e um GPS, cada um destes elementos terá um script Lua próprio associado a ele.

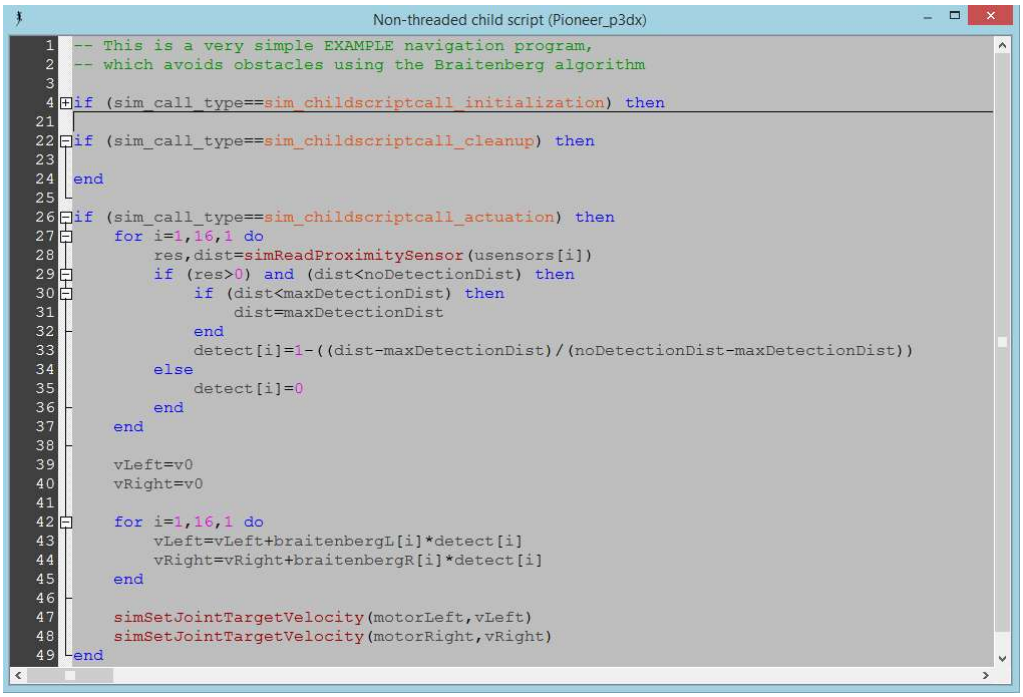


Figura 6.23. Exemplo de *script* em Lua para o controle de um robô Pioneer usando o Sonar/Ultrassom (*usensor*) para gerar comandos de desvio de obstáculos em um robô com cinemática diferencial (controle independente de velocidade dos motores *vLeft* e *vRight*).

Um elemento importante para passar valores (variáveis) de um script para outro em Lua dentro de uma simulação do V-REP é o uso dos *tubes* ou de *signals*. Por exemplo, se desejamos ler os dados de um sensor laser Sick e enviar estes dados para outro script responsável pela ativação dos motores do robô, podemos passar estes dados através de um *communicationTube* ou *Set/Get-Signal*. A Figura 6.24 apresenta um exemplo do uso de signals para comunicar o script do laser com o script do robô.

Um outro elemento importante na interação com o V-REP é a leitura de eventos do teclado, permitindo assim controlar robôs pelo teclado ou ativar algum tipo de funcionalidade durante a execução da simulação. A Figura 6.25 apresenta um exemplo de leitura do teclado, com a exebição das teclas lidas na barra de status. A geração de arquivos de *log* também é uma outra funcionalidade importante para o desenvolvimento de aplicações robóticas, podendo ser implementado de modo direto através de comandos da própria linguagem Lua.

6.4.4.2. API remota

O V-REP permite diferentes modos de programação e comunicação com o simulador. A API remota é um destes modos, e trabalha com o conceito cliente/servidor, do lado do servidor disponibilizando as informações da cena e para o cliente possibilitando seu controle através de chamadas a API. Um arquivo de cena com exemplos dos diferentes modos de programação e comunicação com o V-REP é disponibilizado na pasta “scenes”,

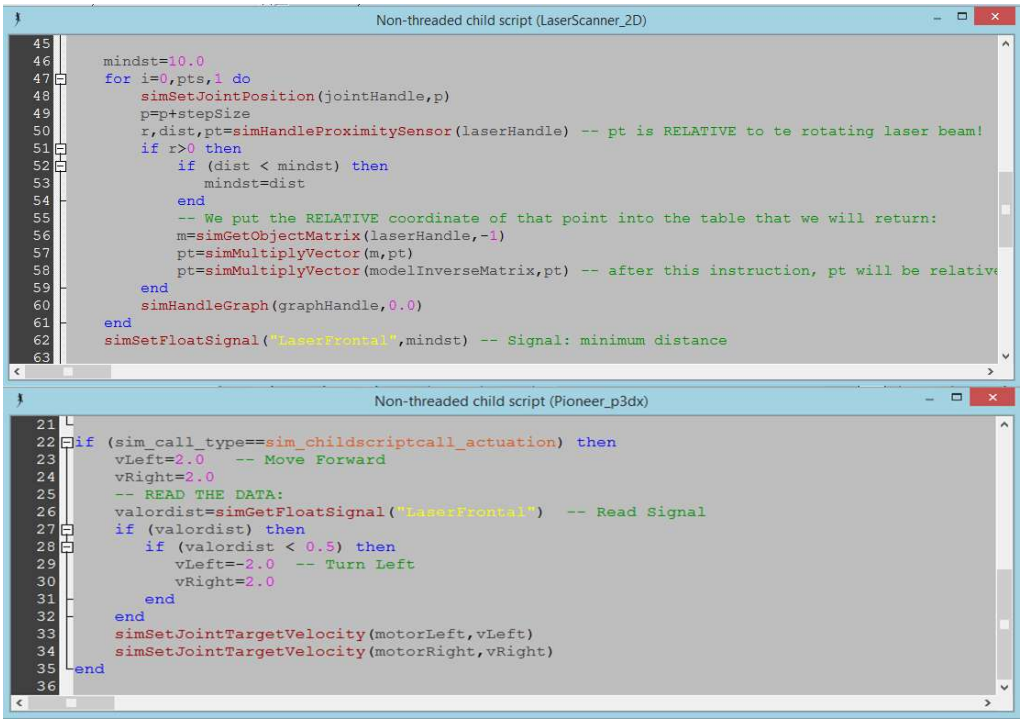


Figura 6.24. Exemplo de *scripts* em Lua com um *signal* para comunicar dados entre os *scripts*.

Figura 6.25. Exemplo de *scripts* em Lua com um a leitura de teclas.

denominado de “controlTypeExamples.ttt”, o qual inclui um exemplo de robô controlado através da API remota.

A primeira etapa para a sua utilização é a habilitação do servidor na simulação. Essa habilitação se dá através do comando *simExtRemoteApiStart(#Porta)* inserido no *script* principal Lua (ver Seção 6.4.4.1) da simulação, na condição de inicialização. A Figura 6.26 mostra um exemplo de como inicializar o servidor da API na porta 19999.

No lado do cliente é preciso acessar as funções da API remota para possibilitar a interação com a simulação. Em C/C++, as funções da API são incluídas com a adição dos

```
if (sim_call_type==sim_mainscriptcall_initialization) then
   simOpenModule(sim_handle_all)
   simHandleGraph(sim_handle_all_except_explicit,0)

   -- habilitando servidor na porta 19999
   simExtRemoteApiStart(19999)
end
```

Figura 6.26. Habilitação do servidor da API remota na simulação do V-REP na porta 19999.

arquivos[13] "extApi.h", "extApi.c", "extApiPlatform.h" e "extApiPlatform.c" ao projeto da aplicação.

Para outras linguagens, é disponibilizada uma biblioteca[14] constituída com base na API de C/C++, possibilitando acesso à simulação, em qualquer linguagem que possa incorporar esta biblioteca.

Cerca de 100 funções estão disponíveis[15] para acesso a simulação. Todas as funções são facilmente reconhecidas pelo prefixo "simx". A função que faz o estabelecimento da conexão com o servidor e, portanto, será sempre a primeira a ser utilizada é a *simxStart*, onde são passados os parâmetros de comunicação, como por exemplo, IP e porta do servidor.

6.4.4.3. V-REP, Matlab e o Robotics Toolbox

O simulador V-REP disponibiliza uma API remota para Matlab, permitindo o desenvolvimento de aplicações cliente para controle de robôs móveis e manipuladores.

Como requisito para utilização do Matlab com V-REP, é necessário que se copie para o diretório em que será executado o programa cliente desenvolvido em Matlab, a biblioteca da API remota do V-REP apropriada para o sistema operacional cliente ("remoteApi.so" para linux, "remoteApi.dll' para windows, ou "remoteApi.dylib" para Mac)[16]. Se for possível escolher entre uma versão 32-bit ou 64-bit deste arquivo, deve-se escolher a versão que corresponde a instalação do Matlab (32-bit ou 64-bit). Além da biblioteca da API remota, também devem ser copiados os arquivos "remAPI.m" e "remoteApiProto.m" diponibilizados pelo V-REP[17]. Um exemplo de como criar uma função em Matlab para acesso a simulação do V-REP também é disponibilizado junto aos arquivos citados acima ("simpleTest.m").

[13] Disponibilizados junto com a instalação do V-REP e acessíveis pelo caminho programming/remoteApi/ (partindo-se da pasta raiz do V-REP).

[14] Deve-se usar a biblioteca da API remota apropriada para o sistema operacional cliente, sendo, "remoteApi.dll" para Windows, "remoteApi.dylib" para Mac e "remoteApi.so" para Linux. Partindo-se da pasta de instalação do V-REP é possível obter a biblioteca em programming/remoteApiBindings/lib/lib/.

[15] A documentação da API remota pode ser obtida no manual do V-REP que está disponível em `http://www.coppeliarobotics.com/helpFiles/`.

[16] Procurar na pasta programming/remoteApiBindings/lib/lib

[17] Procurar na pasta programming/remoteApiBindings/matlab/matlab

A figura 6.27 mostra um exemplo de código em Matlab para iniciar a comunicação entre o cliente e o V-REP, e iniciar a simulação. Antes de iniciar a comunicação com o Matlab é preciso que uma simulação (cena) seja carregada no V-REP e o servidor da API Remota esteja habilitado nesta simulação. O arquivo de cena "controlTypeExamples.ttt", localizado na pasta de "scenes" do V-REP inclui um exemplo de como ativar o servidor da API remota e controlar um robô através desta API remota.

```
vrep=remApi('remoteApi');
id = vrep.simxStart('127.0.0.1', 19999, true, true, 2000, 5);
res = vrep.simxStartSimulation(id, vrep.simx_opmode_oneshot_wait);
```

Figura 6.27. Exemplo de como iniciar a conexão do cliente em Matlab com o simulador V-REP executado na máquina local, IP 127.0.0.1, na porta 19999.

As mesmas funções da API que estão disponíveis para outras linguagens (C/C++, Python, Java e LUA), também estão disponíveis para uso no Matlab, e também possuem o prefixo "simx". Por exemplo, "simxSetJointTargetPosition" e "simxSetJointTargetVelocity" podem ser usadas para definir a posição e velocidade desejada para as juntas de um robô manipulador, e "simxGetObjectPosition" e "simxGetObjectOrientation" para obter a posição e orientação de um objeto, incluindo o efetuador de um robô manipulador.

A *Robotics Toolbox* para Matlab [Corke 2011] já possui um conjunto de funções e algoritmos implementados que auxiliam no estudo e desenvolvimento de sistemas de controle para robôs móveis e manipuladores. Esta toolbox fornece, por exemplo, funções para modelar e simular a cinemática e dinâmica de robôs manipuladores e veículos não-holonômicos, gerar trajetórias e planejar rotas, lidar com transformações homogêneas de sistemas de coordenadas, algoritmos para localização, mapeamento, SLAM (*Simulataneous Localization and Mapping*), dentre outras. O livro [Corke 2011] apresenta diversos exemplos destas funções em Matlab usadas para o desenvolvimento de aplicações e pesquisas em robótica.

Devido a variedade de funções implementadas na *Robotics Toolbox* para Matlab, o uso combinado desta *toolbox* com o simulador V-REP se mostra como uma solução atrativa para a prototipagem rápida de um sistema de controle e percepção para robôs. Naturalmente, estas ferramentas também podem ser utilizadas em cursos de robótica a nivel de graduação e pós-graduação. Tento em vista essa motivação no ensino da robótica, Renaud Detry criou o projeto *TRS: An Open-source Recipe for Teaching/Learning Robotics with a Simulator*, disponível em `http://ulgrobotics.github.io/trs` para um curso na Universidade de Liège. Este projeto de código aberto fornece um ambiente de simulação para V-REP (arquivo ".ttt"), um conjunto básico de funções para Matlab, exemplos de código, e uma estrutura de projeto, com objetivo, documentação e metas intermediárias, para ser executado por alunos de um curso de robótica de nível de mestrado. O TRS foi apresentado em um *workshop* do ICRA[18] em 2014, e um tutorial do TRS foi realizado no IROS[19] em 2014, com previsão de tutoriais a serem apresentados no ICVS[20]

[18]IEEE International Conference on Robotics and Automation

[19]IEEE/RSJ International Conference on Intelligent Robots and Systems

[20]International Conference on Computer Vision Systems

2014, Ro-Man[21] 2015 e IROS 2015. Portanto, é possível notar o crescente interesse da comunidade de robótica para com o simulador V-REP, e em particular, a grande quantidade de possibilidades que são abertas pelo fato de poder conectar o V-REP com este *toolbox* do Matlab.

6.4.4.4. V-REP e ROS (*RObot Operating System*)

O V-REP não poderia deixar de oferecer uma interface para com a plataforma ROS (*RObot Operating System* [22]), uma das ferramentas mais conhecidas e utilizadas por quem desenvolve aplicações com robôs, principalmente com robôs reais. O ROS é um *framework* para o desenvolvimento de aplicações robóticas baseado em uma arquitetura de serviços robóticos, e que muitas vezes é visto como um "sistema operacional robótico", uma vez que provê um *middleware* de acesso a diferentes plataformas robóticas, assim como diversas aplicações de alto-nível (pacotes ROS) que implementam comportamentos e funcionalidades (p.ex. planejamento, auto-localização, navegação) para robôs. O ROS pode ser usado junto com simuladores, como por exemplo o Gazebo (ver 6.3.1 e 6.3.2) e também com o V-REP.

O ROS permite o controle de robôs reais ou simulados via *ROS nodes*. O V-REP oferece uma interface de comunicação com o ROS, que é naturalmente um sistema distribuído orientado a serviços, e assim, é possível comunicar o V-REP com o ROS, através de nodos que realizam trocas de mensagens. O V-REP provê a publicação de mensagens via *V-REP ROS publishers* e a assinatura de mensagens via *V-REP ROS subscribers*, que são mecanismos familiares a quem trabalha com os serviços de publicação e assinatura de mensagens dos nodos do ROS. Entretanto, o próprio ROS e a comunicação com o ROS são mecanismos mais complexos e que requerem um estudo mais aprofundado da interface ROS para o desenvolvimento de aplicações que façam uso desta abordagem.

O arquivo exemplo de cena "controlTypeExamples.ttt" também inclui um exemplo de controle de um robô através do ROS, onde é necessário ter o ROS instalado, configurado e preparado para se comunicar com o robô deste exemplo.

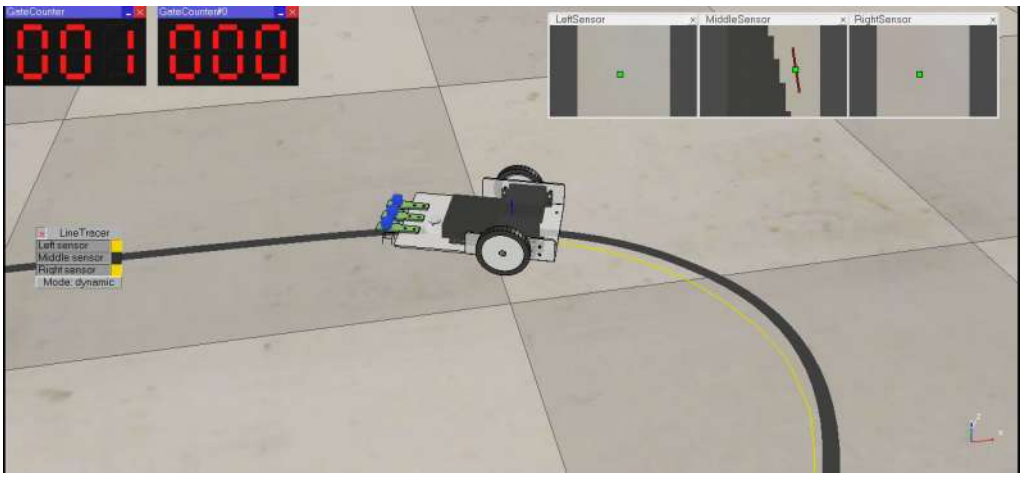
6.5. Aplicações da Robótica Móvel e Inteligente

Nesta seção, são apresentados exemplos de aplicações de robótica móvel e inteligente, relacionando o seu projeto e desenvolvimento, com a possibilidade de realizar trabalhos através da implementação de simulações.

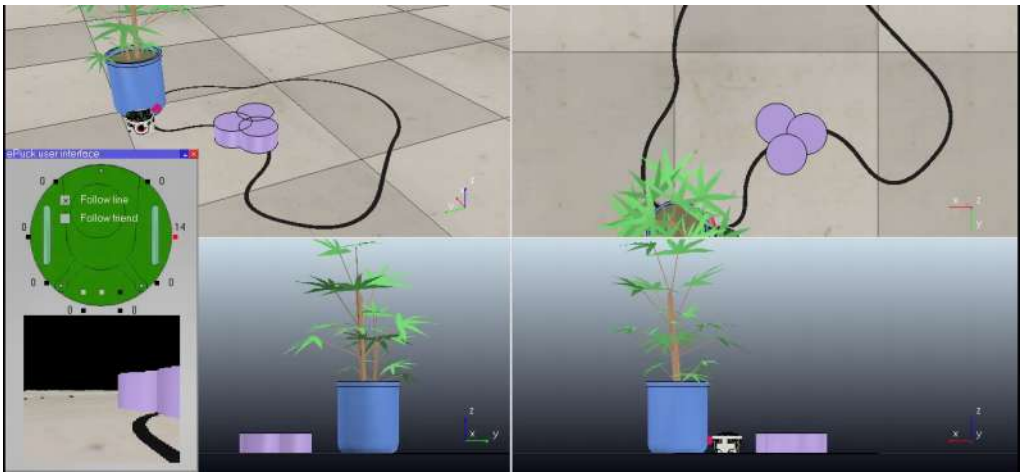
Uma primeira aplicação robótica muito presente em competições de robótica (como a OBR [23], por exemplo), por ser relativamente simples de ser implementada e, também, não muito dispendiosa em termos de sensores, são os robôs seguidores de linhas. As linhas são demarcadas, habitualmente feitas no piso, e servem como rotas pré-planejadas de deslocamento. Os robôs tem um comportamento autônomo (sem intervenção humana), seguindo a linha que define sua trajetória.

[21] International Sysmposium on Robot and Human Interactive Communication
[22] ROS: http://www.ros.org/
[23] OBR - Olimpíada Brasileira de Robótica: http://www.obr.org.br/

Um exemplo de seguidor de linha[24] utilizando V-REP é mostrado na Figura 6.28(a), onde, um robô é equipado com três sensores (câmeras) focados no solo, visando a detecção das linhas. Quando a linha é detectada pelo sensor da esquerda, o robô é rotacionado para a esquerda e o inverso ocorrendo ao ser detectada a linha pelo sensor da direita. O objetivo do robô seguidor de linha é manter a linha (região escura) com a maior área possível junto ao sensor central, onde nessa situação o robô deve seguir em frente.



(a) Seguidor de linha 1 (cena “LineTracer-threaded.ttt”)



(b) Seguidor de Linha 2 (cena “e-puckDemo.ttt”)

Figura 6.28. Exemplos de robôs preparados para seguir linhas.

A Figura 6.28(b) mostra um segundo exemplo de robô seguidor de linha[25], um pouco mais complexo. Este robô é capaz de contornar obstáculos (usando um sensor de proximidade) que estejam sobre as linhas, tornando-se, portanto, mais robusto para uma aplicação real. Neste exemplo ainda, mostra-se a cena de quatro pontos de vistas diferentes.

[24] O vídeo da simulação no V-REP do seguidor de linhas 1 está disponível em `https://youtu.be/ij_UVMWpQIs`.

[25] O vídeo do seguidor de linhas 2 está disponível no endereço `https://youtu.be/wf9n1PrRvW4`.

Um robô seguidor de linha é, portanto, um robô estritamente reativo que é capaz de caminhar por rotas previamente definidas. Robôs seguidores de linhas são comuns em competições robóticas, mas também existem produtos comerciais baseados neste tipo de robôs, como o OZOBOT [Evollve 2015]. Inclusive alguns robôs móveis industriais, como os AGVs (*Automated Guided Vehicles*) da Kiva Systems [Kiva Systems 2015], são seguidores de linhas que foram adotados pela Amazon para a distribuição de mercadorias em seus armazéns.

Outro comportamento reativo similar ao seguidor de linhas é o seguidor de paredes. Com essa estratégia usualmente é possível percorrer todos os cômodos de um ambiente fechado, por isso pode ser empregado para robôs que efetuem o mapeamento de ambientes. Um exemplo de comportamento de seguidor de paredes[26] é mostrado na Figura 6.29. Neste exemplo, um robô Pioneer P3dx [Adept MobileRobots 2015] com GPS, um sensor *laser* Sick [SICK 2015] e três câmeras desempenha comportamentos de seguidor de paredes (Sick) e Seguidor de linhas (câmeras e Sick para desvio de obstáculos). O robô inicia em um ambiente interno onde segue as paredes utilizando seu sensor Sick para manter uma distância mínima e máxima da parede (objeto) e verificando paredes e objetos a frente. Este comportamento permanece até encontrar a porta de saída para o ambiente externo, passando assim, a ser um robô seguidor de linhas. O objetivo dessa simulação é fazer com que o robô alcance uma coordenada específica do ambiente (onde uma pessoa está sentada aguardando sua chegada) usando somente a combinação de comportamentos reativos: seguir paredes, desviar de obstáculos, seguir linhas, seguir em direção a uma coordenada alvo.

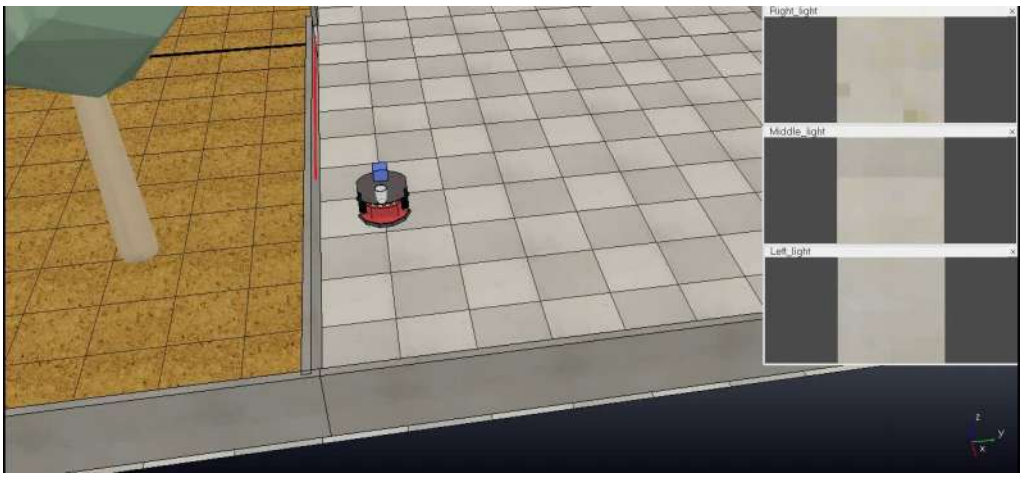


Figura 6.29. Robô que executa comportamentos reativos: seguir paredes e linhas.

Pode ser interessante para um robô desempenhar um comportamento de perseguição, ou seja, seguir algum outro robô ou objeto em deslocamento. Esse comportamento é reativo também. Na Figura 6.30, mostra-se um exemplo de simulação em que um robô hexapod[27] (robô de seis "pernas") branco persegue um outro hexapod vermelho utilizando

[26]O vídeo do seguidor de paredes e linhas está disponível no endereço `https://youtu.be/-1KCFHhnSjA`.

[27]O vídeo da simulação de robô perseguidor no V-REP está disponível em `https://youtu.be/zn9KSZZNenI`.

uma câmera. O perseguidor procura no ambiente pela cor vermelha em uma tonalidade e brilho específicos e busca manter a região encontrada dentro de certas dimensões e centralizada. Portanto, caminha à frente sempre que nota que a largura da região segmentada do robô perseguido é menor que certo valor; caminha para trás quando percebe que é maior que o valor máximo aceito; e busca também deixar o centro de massa da região segmentada próximo ao centro horizontal da imagem fornecida pela câmera.

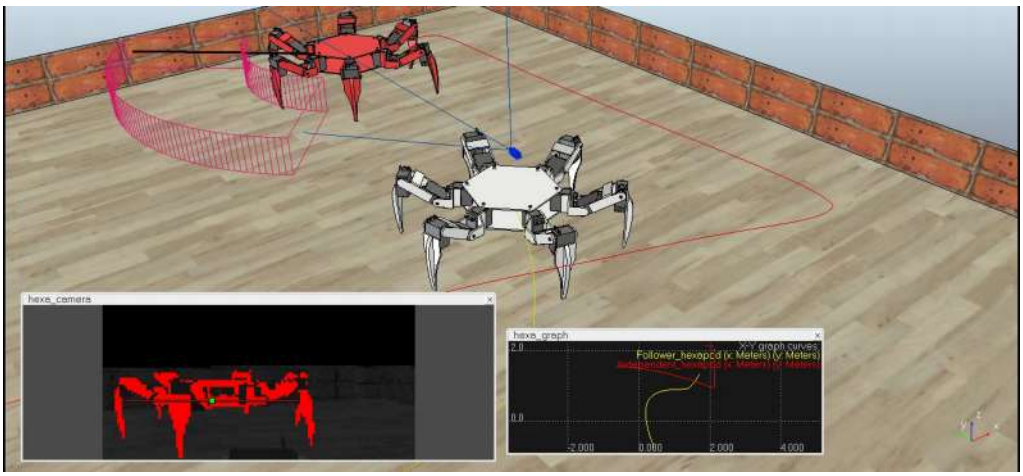


Figura 6.30. Robô que efetua o comportamento reativo de perseguidor utilizando a cor, sendo, o robô vermelho perseguido pelo branco (cena "imageProcessingExample.ttt").

É possível também se obter o comportamento adequado de um robô utilizando um algoritmo evolutivo de seleção natural. Na Figura 6.31, é mostrado um exemplo desse tipo de abordagem[28], onde 5 indivíduos são mantidos em simulação a cada instante. Quando o V-REP detecta que o robô colidiu ele é eliminado e um outro indivíduo é gerado com base nos melhores da população, sendo os melhores aqueles que mais se locomovem pelo ambiente sem colisões. O mais interessante é que o robô não é programado manualmente para reagir às informações obtidas pelos sensores, e sim, à medida que as gerações passam, um comportamento interessante emerge sendo obtido pela evolução dos robôs simulados. A obtenção de comportamentos por evolução e seleção natural só se torna viável em ambientes simulados fiéis (com um motor físico consistente, robôs e sensores fiéis à realidade), e com um grande número de robôs e/ou repetições. Só assim, após um longo processo de "evolução" (simulada) este comportamento poderá ser confiável suficiente para ser utilizado em um ambiente real.

O mapeamento de ambientes é um outro problema importante na robótica. Este mapeamento permite que um sistema robótico crie um mapa e depois planeje suas ações com base em uma descrição fidedigna do ambiente gerada neste processo. Com o V-REP pode-se desenvolver e aprimorar algoritmos de mapeamento[29]. A Figura 6.32 mostra a criação de uma nuvem de pontos descrevendo o ambiente (mapa 3D) utilizando um sensor

[28]O vídeo que mostra a utilização da seleção natural está disponível em `https://youtu.be/LTOnQYS60Fo`.

[29]O vídeo de um exemplo de mapeamento 3D de um ambiente e está disponível no endereço `https://youtu.be/IP14qvYzqtw`.

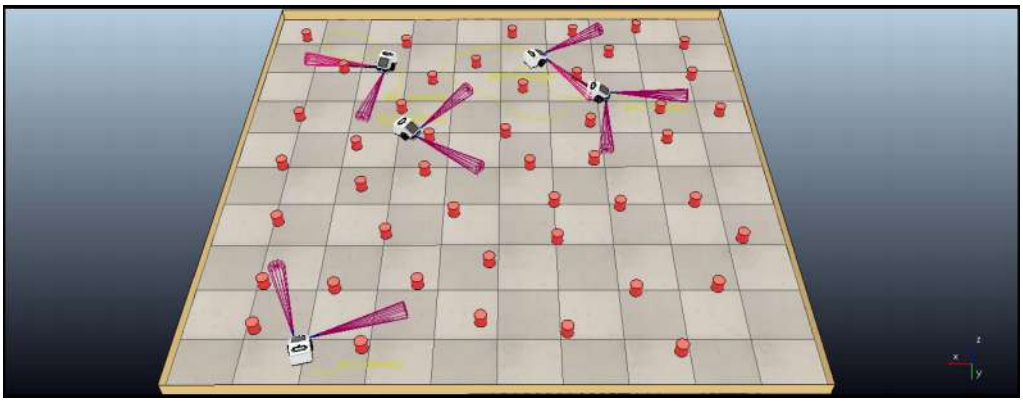


Figura 6.31. Utilizando o V-REP e algoritmo de seleção natural para encontrar comportamento adequado para robôs (cena "naturalSelectionAlgo.ttt").

Hokuyo [Hokuyo Automatic 2015] sendo rotacionado e inclinado por uma base (*pan/tilt*), que, por sua vez, está posicionada sobre um robô móvel com esteiras. A nuvem de pontos gerada é exibida em azul na janela superior direita da tela, enquanto a propagação do *laser* do Hokuyo é mostrada em vermelho na cena. O sensor Hokuyo é 2D (mapeia um corte planar da cena), por isso torna-se necessária sua inclinação e giro para possibilitar a captura do mapa 3D da cena. A grande vantagem de se usar um sensor como o Hokuyo nesta tarefa é seu preço, ou seja, caso o algoritmo seja utilizado em um ambiente real, o custo de constituição desse sistema seria menor do que a aquisição de um sensor *laser* 3D, como por exemplo, o Velodyne [Velodyne LiDAR 2015].

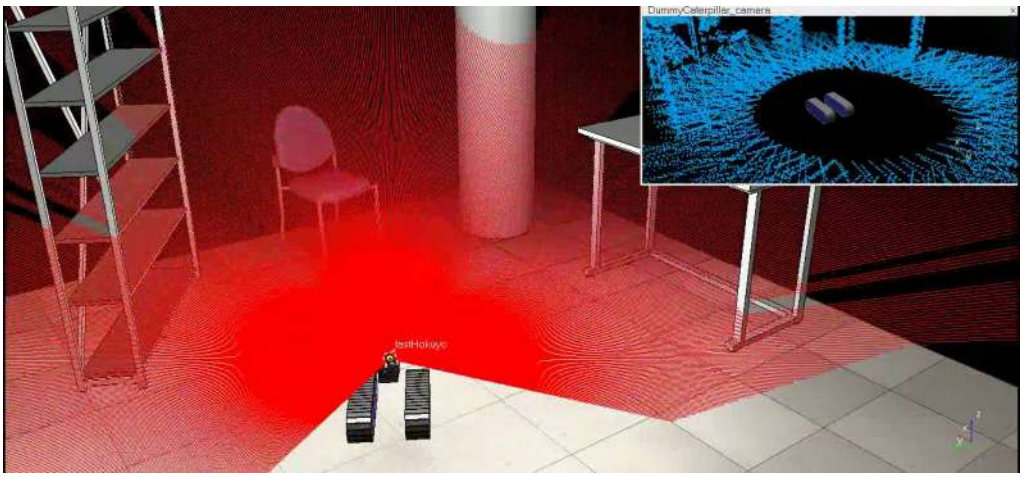
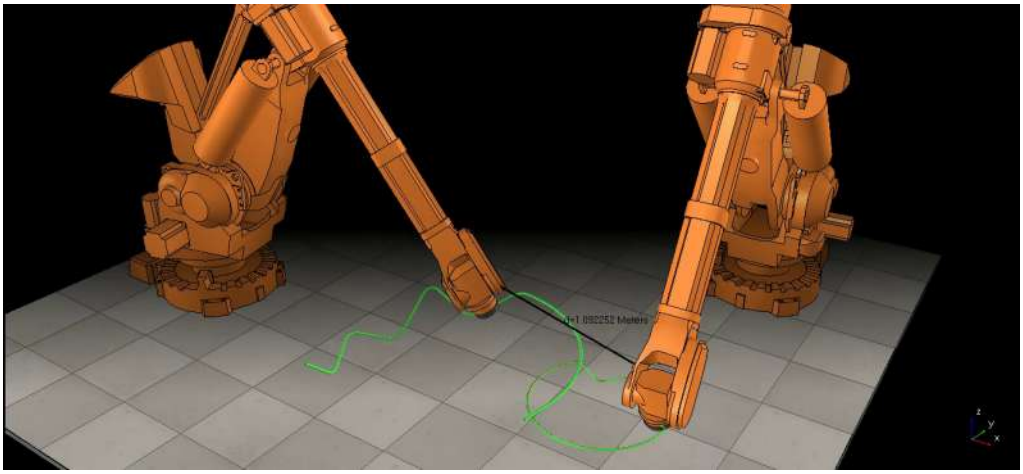


Figura 6.32. Mapeamento 3D do ambiente utilizando um Hokuyo e um suporte *Pan/Tilt* (cena "environmentMapping.ttt").

O V-REP também pode ser útil para aplicações industriais que utilizem braços robóticos. Pode-se testar rotas, prevenindo-se de eventuais colisões, antes da utilização na prática em um ambiente real. O V-REP possui algumas rotinas próprias de detecção de colisões, distância entre objetos, formando um arcabouço matemático eficaz e de fácil implementação. A Figura 6.33(a) apresenta um teste efetuado com dois braços robóticos[30]

[30]A execução do teste é mostrado no endereço `https://youtu.be/Q8kgWcxYaY8`.

que percorrem suas rotas (em verde) e a distância mínima entre eles é monitorada. Já a Figura 6.33(b) apresenta dois braços robóticos[31] efetuando movimentos planejados e sincronizados no desempenho da tarefas de mover bolinhas de um copo a outro, ou mesmo, empilhar copos.



(a) Planejamento e teste de Rotas (cena "2IndustrialRobots.ttt")

(b) Planejamento de movimentos (cena "motionPlanningAndGraspingDemo.ttt")

Figura 6.33. V-REP no planejamento e teste de rotas/movimentos com braços robóticos.

Outra funcionalidade interessante que o V-REP possibilita é a retirado (corte) de material, em que os cortes se dão apenas em alguns objetos especiais declarados com a propriedade cortável (ver Seção 6.4.2.2). É possível, portanto, colocar uma fresa acoplada a um braço robótico[32] e simular a sua atuação no corte do material, buscando assim sanar eventuais problemas na programação dos movimentos do braço robótico e, ainda, visualizando o resultado final do corte. Na Figura 6.34, a simulação da usinagem de uma peça de aço é mostrada.

[31] O vídeo que mostra a exibição da simulação dos movimentos dos braços robóticos está disponível em `https://youtu.be/BI6v-uRnbuc`.

[32] Vídeo exemplo de braço robótico com fresa no V-REP está disponível em `https://youtu.be/4aEhAd5EhbQ`.

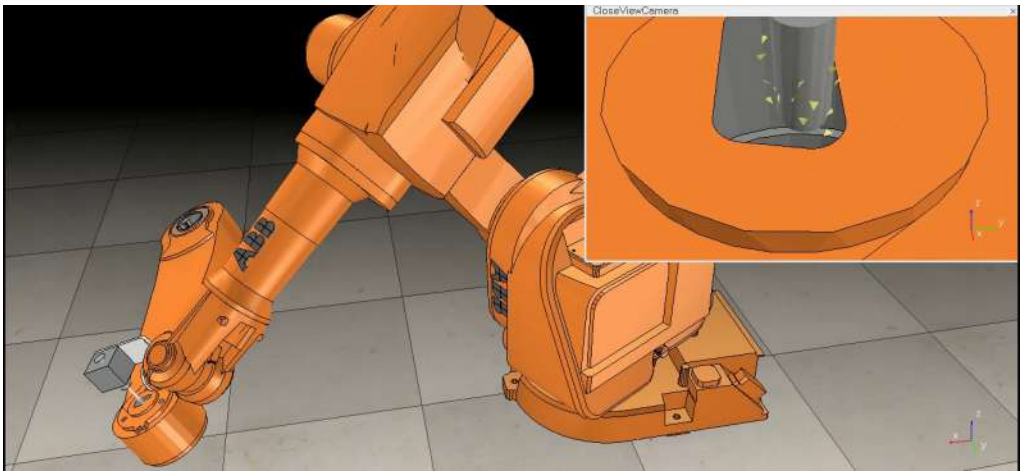


Figura 6.34. Braço robótico com fresa efetuando os cortes programados no material (cena "millingRobot.ttt").

De maneira similar ao corte, o V-REP permite o depósito de materiais sobre os objetos da cena. Pode-se então, efetuar uma simulação da soldagem industrial usando braços robóticos[33]. A Figura 6.35 mostra uma tela exemplo de uma operação de solda efetuada por um braço robótico no V-REP.

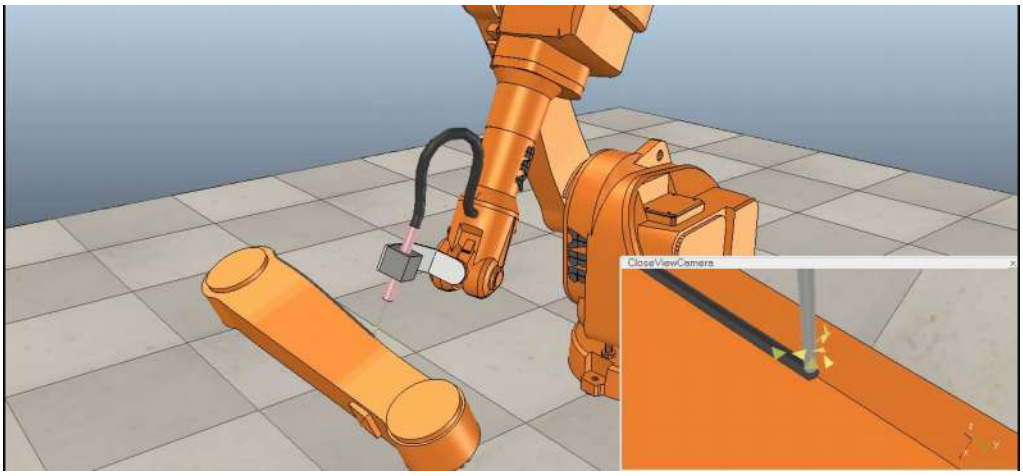


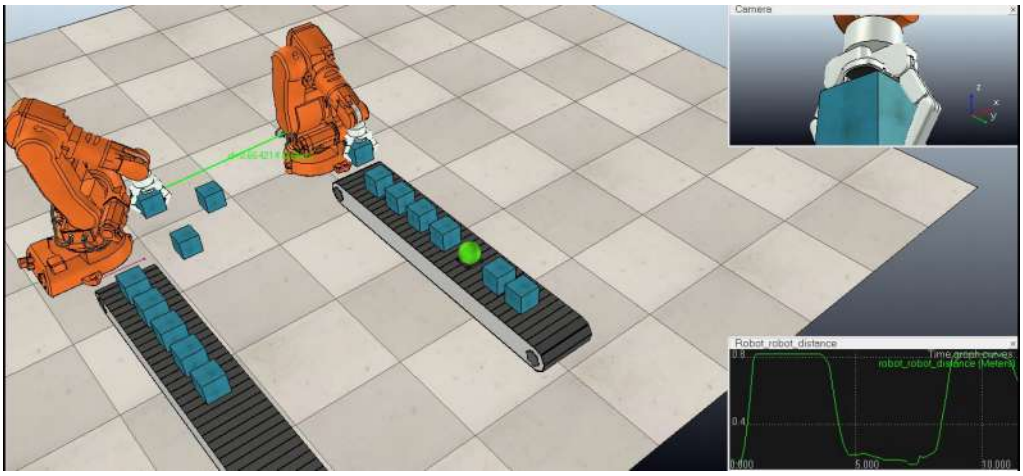
Figura 6.35. Braço robótico efetuado a solda em uma peça (cena "weldingRobot.ttt").

Além de robôs e sensores, o V-REP possui alguns modelos já prontos que auxiliam na simulação de operações do cotidiano de parques fabris. Por exemplo, estão disponíveis modelos de esteiras que transportam mercadorias por fábricas. Atrelado a modelos de braços robóticos pode-se simular processos produtivos, ou mesmo, uma unidade de despacho de encomendas. Na Figura 6.36(a) é mostrada uma simulação com esteiras e robôs de base fixa[34] preparados para organizar caixas (despacho de mercado-

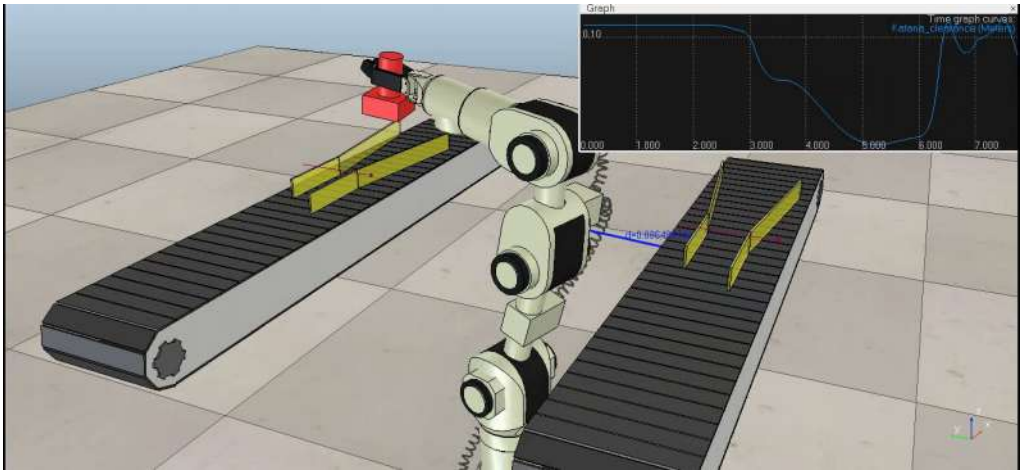
[33]Vídeo exemplo do braço robótico soldador em V-REP que está disponível em `https://youtu.be/zvUt0mqfGpM`.

[34]Vídeo exemplo do braço robótico e esteiras organizando caixas está disponível em `https://youtu.be/7-l5euZsHfE`.

rias). Na Figura 6.36(b) é apresentado um exemplo de cena, onde, esteiras de montagem são realimentadas por um braço robótico[35].



(a) Esteiras para organização de despacho de caixas (cena "BarrettHandPickAndPlace.ttt")



(b) Esteira de montagem (cena "katanaRobotWithCableSimulation.ttt")

Figura 6.36. Exemplos de simulação de esteiras industriais.

Além de braços robóticos articulados convencionais, estão disponíveis no V-REP robôs paralelos (*parallel manipulators*) específicos para manipulação em ambientes industriais. Estes robôs permitem uma movimentação de mercadorias mais eficiente que braços robóticos. Esses robôs podem ser afixados sobre as esteiras, possibilitando a automação da retirada de objetos em uma linha de produção. A Figura 6.37 apresenta um exemplo de linha industrial[36], onde, as mercadorias são destinadas segundo sua cor e forma. Esta simulação emprega dois robôs manipuladores IRB-360 [ABB 2015b], uma esteira e uma câmera acoplada no início da esteira para a obtenção da posição, forma e cor das peças.

[35]No endereço `https://youtu.be/QE4GC4L1ioI` pode-se ver o vídeo gerado da simulação de esteiras de montagens alimentadas por braço robótico.

[36]Está disponível o vídeo da simulação da esteira e os robôs manipuladores no endereço `https://youtu.be/iXDJ-fSr9Uw`.

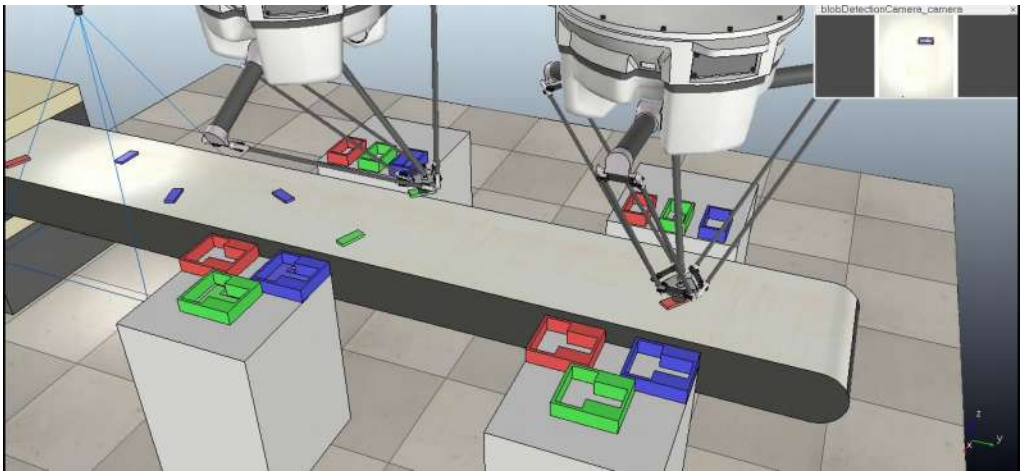


Figura 6.37. Exemplo de robôs manipuladores que atuam sobre uma esteira, fazendo a destinação de objetos segundo forma e cor (cena "blobDetectionWithPickAndPlace.ttt").

Além destes exemplos, é importante destacar que o V-REP também oferece modelos de diferentes tipos de veículos que podem ser acessados através do menu na opção *Load-Model* ou no *browser* de modelos. Estes modelos incluem robôs terrestres, na pasta *vehicles*, com modelos de carros (manta.ttm), de uma escavadeira (excavator.ttm), de uma motocicleta (motorbike.ttm) e por fim de um veículo com cinemática Ackermann na pasta *examples* (SimpleAckermannsterring.ttm). Além dos veículos terrestres também existem modelos de veículos aéreos, notadamente, de um helicóptero (pasta *vehicles*: helicopter.ttm) e de um quadcóptero (pasta *robots-mobile*: Quadricopter.ttm).

6.6. Robótica Móvel e Inteligente: Perspectivas futuras

Atualmente os robôs móveis e articulados já podem ser considerados "robôs inteligentes", pois possuem a capacidade de: (*i*) perceber, decidir e agir de forma autônoma; (*ii*) conduzir veículos de modo autônomo; (*iii*) transportar e entregar mercadorias usando veículos aéreos; e inclusive, (*iv*) executar atividades humanas como caminhar, manipular e transportar objetos (robôs humanóides). Os robôs inteligentes tem sido aplicados na indústria (p.ex. braços industriais e manipuladores, AGVs), em ambientes domésticos (p.ex. robôs aspiradores de pó, cortadores de grama, limpadores de piscina), em atividades comerciais (p.ex. taxis robóticos, garçons automatizados), no entretenimento e na pesquisa (p.ex. competições robóticas, animais robóticos como cães e dinossauros, robôs para crianças e cuidadores de idosos), no ensino (p.ex. robôs usados no ensino de programação, robôs LEGO e NAO), em atividades que sejam arriscadas ou perigosas para seres humanos (p.ex. pulverização de agrotóxicos, colheita automatizada, exploração de outros planetas, mineração, busca e salvamento em desastres), no ar-terra-água (p.ex. robôs de exploração e monitoramento), e inclusive em atividades bélicas, com os Drones, sentinelas e outros robôs de combate. Podemos concluir que os robôs já estão plenamente disseminados em nossa sociedade atual e estarão cada vez mais presentes em nosso dia-a-dia, afinal quem de nós não possui um produto que foi manufaturado por um robô, como um *smartphone* ou mesmo um carro.

Neste contexto, é de grande importância que a sociedade atual esteja preparada para este futuro não muito distante, onde vamos conviver diariamente com robôs, e principalmente, precisamos preparar as gerações futuras para que estejam prontas para conviver, trabalhar e desenvolver novos projetos na área robótica. Além disto, devemos estar preparados para este futuro que nos aguarda, através da discussão sobre os aspectos legais e éticos que a robótica irá impor a sociedade, além de estar preparados para os impactos que esta automação e a "invasão do robôs" irá ter em nossa sociedade.

Muitas pessoas tem uma visão fantasiosa da robótica e do futuro, influenciadas pela ficção científica (que nem sempre reflete a realidade e uma expectativa futura mais realista). Por exemplo, um grande número de pessoas acredita que os robôs podem "roubar"nossos empregos e que serão os responsáveis pelo desemprego em massa no futuro, mas será que isto é verdade? Existem diversos estudos que demonstram que os robôs irão inclusive resolver enormes problemas dos humanos, como por exemplo, na produção de alimentos, onde a automação agrícola é indispensável para uma produção suficiente de alimentos para a atual população de nosso planeta. Outras tarefas como cuidar de idosos será muito em breve de grande importância, considerando o envelhecimento da população e o aumento da expectativa de vida. E afinal, quando aumentamos a produção e a produtividade (otimização da produção), isto não gera mais empregos? Esta é uma discussão importante, onde devemos buscar uma desmistificação da robótica, para que possamos aproveitar melhor os seus benefícios. Mas é claro que devemos nos preparar para os impactos que a automação pode trazer, pois quem sabe, algumas profissões podem mesmo ter que mudar, como a de motorista de táxi ou de garçom/entregador de mercadorias. De qualquer modo, não podemos ficar a margem desta grande revolução que é a inserção da robótica junto a sociedade moderna.

Em relação as questões legais e éticas, considerando a capacidade de um robô poder dirigir um carro sem a necessidade da supervisão constante de uma pessoa (indicada no item (*ii* acima), um dos importantes questionamentos que surgem é: O que acontece em caso de acidentes? Quem é o responsável legal, uma vez que não havia uma pessoa conduzindo o veículo? Estes questionamentos demonstram claramente que o desenvolvimento de aplicações robóticas vai além das questões técnico-científicas, alcançando inclusive outras áreas como o Direito. Será necessária uma colaboração direta entre pesquisadores e experts em robótica, juntamente com juristas, a fim de estabelecer novas regras e leis para este tipo de situações. Neste caso em particular, acreditamos que também será necessário desenvolver equipamentos específicos, como caixas pretas que registrem tudo o que se passa dentro e fora (sensores) do carro, bem como serão necessárias leis específicas que obriguem a adoção de tais equipamentos tal como ocorre em aviões. Somente com tais equipamentos e com o registro detalhado da cena do acidente, poderemos determinar os responsáveis e penalizá-los legalmente, assim como será possível incrementar a segurança destes equipamentos. Isto demonstra quanto trabalho e a necessidade URGENTE de formar pessoal capacitado para tratar de questões como estas, seja no exterior ou seja em nosso país.

Considerando a capacidade de um robô aéreo aéreo indicada no item *iii*, onde um veículo aéreo não tripulado pode também cair, provocando danos materiais e pessoais (em alguns países drones acima de um determinado tamanho já são proibidos e/ou plenamente regulamentados em termos de seu uso). Constata-se, assim como para os veículos

terrestres, que são necessários estudos e medidas em relação a segurança, mas também em relação a privacidade, uma vez que tais veículos (Drones, VANTs, UAVs) têm sido amplamente utilizados para realizar filmagens, invadindo casas e a privacidade das pessoas. E mais grave ainda, é que usualmente as pessoas não tem como impedir ou reagir, pois como evitar que um drone invada um espaço privado? Questões legais/éticas devem ser discutidas, e novamente, devem ser buscadas soluções não somente legais, mas também tecnológicas: quem sabe com o desenvolvimento de uma esquadra de uma esquadra com guardas-drones (*drone cops*) que irão "caçar" os drones ilegais. Isto envolve novamente a necessidade de formação e preparação de pessoal capacitado a desenvolver tais tecnologias e regulamentações.

Por fim, os robôs podem ser usados inclusive em atividades bélicas (e não somente os robôs humanóides, mas também os robôs terrestres, aéreos ou aquáticos). Isto pode ter consequências bastante perigosas para a sociedade, pois mesmo as guerras possuem suas regras, e fica aqui a pergunta: quem já viu um robô de guerra/ataque reconhecer e aceitar um pedido de rendição (bandeira branca) por parte de um humano?

Portanto, o conhecimento, o domínio da tecnologia, a regulamentação da robótica e a discussão de questões éticas/legais, são de grande importância na atualidade em relação a este tema. O Brasil, diante deste quadro, deve buscar um maior investimento e formação de pessoal na área de robótica, para que possamos estar preparados para estas situações com as quais muito em breve seremos confrontados. Em função disto, acreditamos que este mini-curso, através dos conceitos e ferramentas (V-REP) apresentados, tem uma grande relevância, por permitir um maior acesso as tecnologias e conhecimentos ligados a robótica, e assim, fomentar do desenvolvimento desta área estratégica no país.

Referências

[ABB 2015a] ABB (2015a). Robô irb 140. Acessado: 19.03.2015.

[ABB 2015b] ABB (2015b). Robô irb 360. Acessado: 19.03.2015.

[Adept MobileRobots 2015] Adept MobileRobots, I. (2015). Robô pioneer p3dx. Acessado: 19.03.2015.

[Aldebaran Robotics 2015] Aldebaran Robotics, I. (2015). Robô humanóide nao. Acessado: 19.03.2015.

[Arkin 1989] Arkin, R. C. (1989). Towards the unification of navigational planning and reactive control. In *In AAAI Spring Symposium on Robot Navigation*, pages 1–5.

[Arkin and Balch 1997] Arkin, R. C. and Balch, T. (1997). Aura: Principles and practice in review. *Journal of Experimental and Theoretical Artificial Intelligence*, 9:175–189.

[Bradski and Kaehler 2008] Bradski, G. and Kaehler, A. (2008). *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, Inc.

[Cook et al. 2014] Cook, D., Vardy, A., and Lewis, R. (2014). A survey of auv and robot simulators for multi-vehicle operations. In *Autonomous Underwater Vehicles (AUV), 2014 IEEE/OES*, pages 1–8.

[Coppelia Robotics 2015] Coppelia Robotics, G. (2015). Virtual robot experimentation platform - user manual. http://www.coppeliarobotics.com/helpFiles/. Acessado em: 19.03.2015.

[Corke 2011] Corke, P. I. (2011). *Robotics, Vision & Control: Fundamental Algorithms in Matlab*. Springer.

[Correa 2013] Correa, D. S. O. (2013). *Navegação autônoma de robôs móveis e detecção de intrusos em ambientes internos utilizando sensores 2D e 3D*. USP ICMC - Dissertação de Mestrado, Sao Carlos, SP, Brasil.

[Craighead et al. 2008] Craighead, J., Gutierrez, R., Burke, J., and Murphy, R. (2008). Validating the search and rescue game environment as a robot simulator by performing a simulated anomaly detection task. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2289–2295.

[Cytron Technologies 2015] Cytron Technologies, I. (2015). Sonar hc-sr04. http://cytron.com.my/p-sn-hc-sr04. Acessado em: 19.03.2015.

[Department 2014] Department, I. S. (2014). World robotics 2014 - industrial robots and service robots (executive summary wr-2014). http://www.worldrobotics.org/uploads/media/Executive_Summary_WR_2014_02.pdf. Acessado em: 08.05.2015.

[Dudek and Jenkin 2000] Dudek, G. and Jenkin, M. (2000). *Computational Principles of Mobile Robotics*. Cambridge University Press, Cambridge, UK.

[E. Rohmer 2013] E. Rohmer, S. P. N. Singh, M. F. (2013). V-rep: a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*.

[Echeverria et al. 2011] Echeverria, G., Lassabe, N., Degroote, A., and Lemaignan, S. (2011). Modular open robots simulation engine: Morse. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 46–51.

[Evollve 2015] Evollve, I. (2015). Ozobot. http://www.ozobot.com/. Acessado em: 19.03.2015.

[Fernandes et al. 2014] Fernandes, L. C., Souza, J. R., Pessin, G., Shinzato, P. Y., Sales, D., Mendes, C., Prado, M., Klaser, R., Magalhães, A. C., Hata, A., et al. (2014). Carina intelligent robotic car: Architectural design and applications. *Journal of Systems Architecture*, 60(4):372–392.

[Freescale Semiconductor 2015] Freescale Semiconductor, I. (2015). Acelerômetro mma7361l. http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MMA7361LC&tid=vanRED. Acessado em: 19.03.2015.

[Gates 2007] Gates, B. (2007). A robot in every home. *Scientific American*, 296(1):58–65.

[Gerkey et al. 2003] Gerkey, B., Vaughan, R. T., and Howard, A. (2003). The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th international conference on advanced robotics*, volume 1, pages 317–323.

[Grassi Jr. and Okamoto Jr. 2014] Grassi Jr., V. and Okamoto Jr., J. (2014). Arquiteturas de controle: tipos e conceitos. In Romero, R. A. F., Prestes, E., Osório, F., and Wolf, D., editors, *Robótica Móvel*, chapter 4, pages 47–60. Editora LTC, Rio de Janeiro.

[Guizzo 2012] Guizzo, E. (2012). Amazon acquires kiva systems for $775 million. `http://spectrum.ieee.org/automaton/robotics/industrial-robots/amazon-acquires-kiva-systems-for-775-million`. Acessado em: 19.03.2015.

[Harris and Conrad 2011] Harris, A. and Conrad, J. M. (2011). Survey of popular robotics simulators, frameworks, and toolkits. In *Southeastcon, 2011 Proceedings of IEEE*, pages 243–249.

[HiBot 2015] HiBot (2015). Robô serpente acm-r5. Acessado: 19.03.2015.

[Hokuyo Automatic 2015] Hokuyo Automatic, C. (2015). Urg-04lx-ug01. `https://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx_ug01.html`. Acessado em: 19.03.2015.

[InvenSense 2015] InvenSense, I. (2015). Imu mma7361l. `http://www.invensense.com/mems/gyro/mpu6050.html`. Acessado em: 19.03.2015.

[iRobot 2015] iRobot (2015). Roomba. `http://www.irobot.com/For-the-Home/`. Acessado em: 19.03.2015.

[Iyengar and Elfes 1991] Iyengar, S. S. and Elfes, A. (1991). *Autonomous Mobile Robots: Perception, mapping, and navigation*. Autonomous Mobile Robots. IEEE Computer Society Press.

[Jung et al. 2005] Jung, C. R., Osorio, F. S., Kelber, C., and Heinen, F. (2005). Computação embarcada: Projeto e implementação de veículos autônomos inteligente. In *Anais do CSBC'05 XXIV Jornada de Atualização em Informática (JAI)*, chapter 4, pages 1358–1406. SBC, São Leopoldo, RS.

[Kiva Systems 2015] Kiva Systems, L. (2015). Automated guided vehicles. `http://www.kivasystems.com/`. Acessado em: 19.03.2015.

[Koenig and Howard 2004] Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154.

[Landroid 2015] Landroid (2015). Wg794e. `http://www.worxlandroid.com/`. Acessado em: 19.03.2015.

[Mataric 1992] Mataric, M. (1992). Behavior-based control: Main properties and implications. In *In Proceedings, IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems*, pages 46–54.

[Mataric 2014] Mataric, M. (2014). *Introdução a Robótica*. Blucher/UNESP Editora, São Paulo, SP, Brasil, 1ed(traduzido) edition.

[Metta et al. 2006] Metta, G., Fitzpatrick, P., and Natale, L. (2006). Yarp: yet another robot platform. *International Journal on Advanced Robotics Systems*, 3(1):43–48.

[Murphy 2000] Murphy, R. R. (2000). *Introduction to AI Robotics*. MIT Press, Cambridge, MA, USA, 1st edition.

[NASA 2015] NASA (2015). Robonaut r2. `http://robonaut.jsc.nasa.gov/`. Acessado em: 19.03.2015.

[Oxford Mobile Robotics 2015] Oxford Mobile Robotics, G. (2015). Moos. `http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php/Main/HomePage`. Acessado em: 28.04.2015.

[Pocolibs 2015] Pocolibs (2015). `https://www.openrobots.org/wiki/pocolibs/`. Acessado em: 28.04.2015.

[Point Grey Research 2015] Point Grey Research, I. (2015). Point grey chameleon. `http://www.ptgrey.com/`. Acessado em: 19.03.2015.

[PUC-Rio 2015] PUC-Rio, G. (2015). The programming language lua. `http://www.lua.org/`. Acessado em: 19.03.2015.

[Quigley et al. 2009] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5.

[Romero et al. 2014a] Romero, R. A. F., Prestes, E., Osório, F., and Wolf, D. (2014a). Introdução à Robótica Móvel. In *Robótica Móvel*, chapter 1, pages 1–12. Editora LTC, Rio de Janeiro.

[Romero et al. 2014b] Romero, R. A. F., Prestes, E., Osório, F., and Wolf, D. (2014b). Sensores e Atuadores. In *Robótica Móvel*, chapter 2, pages 13–25. Editora LTC, Rio de Janeiro.

[SICK 2015] SICK, A. (2015). Lms 200. `http://www.sick.com/`. Acessado em: 19.03.2015.

[Siegwart et al. 2011] Siegwart, R., Nourbakhsh, I. R., and Scaramuzza, D. (2011). *Introduction to Autonomous Mobile Robots*. The MIT Press, 2nd edition.

[Staranowicz and Mariottini 2011] Staranowicz, A. and Mariottini, G. L. (2011). A survey and comparison of commercial and open-source robotic simulator software. In *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '11, pages 56:1–56:8, New York, NY, USA. ACM.

[Vaughan et al. 2003] Vaughan, R. T., Gerkey, B. P., and Howard, A. (2003). On device abstractions for portable, reusable robot code. In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 3, pages 2421–2427.

[Velodyne LiDAR 2015] Velodyne LiDAR, I. (2015). Velodyne hdl-32e. `http://velodynelidar.com/`. Acessado em: 19.03.2015.

[Vieira and Roqueiro 2014] Vieira, R. d. S. and Roqueiro, N. (2014). Cinemática e dinâmica de robôs móveis. In Romero, R. A. F., Prestes, E., Osório, F., and Wolf, D., editors, *Robótica Móvel*, chapter 3, pages 26–46. Editora LTC, Rio de Janeiro.

[Wolf et al. 2009] Wolf, D., Simoes, E., Osorio, F., and Trindade Junior, O. (2009). Robótica inteligente: Da simulação as aplicações no mundo real. In *Anais do CSBC'09 XXIV Jornada de Atualização em Informática (JAI)*, chapter 6, pages 279–330. SBC, Bento Gonçalves, RS.