知能情報システム創成
DARwIn-OP を使ったロボットプログラミング

立命館大学 情報理工学部 知能情報学科

平成 30 年 4 月 18 日

目 次

1　テーマ概要

ロボットはCGやゲームと異なりその身体性を用いて実世界に直接働きかけることができる。実世界の環境をセンサーで検知し、その状況に応じて対応を適応的に変えることが知能的なロボットを実現する上で必要不可欠である。そこで、本テーマでは7回の授業を使ってヒューマノイド型のロボットを制御するプログラムを製作することを通して、実世界で適応的に動作するロボットを実際に動かして、ロボット制御の基礎を学ぶ。例としてロボット同士でボールを落としあう簡単な対戦動作をさせることを目標とする。

2　内容

2.1　課題

0. シミュレータの起動とサンプルのコンパイル及び実行。シミュレータ内でのロボット、物体操作と実機への転送 (1 週目)
1. 両腕のマスタースレーブ制御の実行と改造によるモーター制御法の習熟 (1-2 週目)
2. 首ふり (2 自由度) の制御プログラムの実装 (2 週目)
3. 画像センサをつかったカラーボールの検出と LED による状態表示 (2-3 週目)
4. ボールを首振りで追跡する制御プログラムの実装 (4 週目)
5. 自分のボールを相手から守りつつ、相手方ロボットのボールを発見し、手で落とす動作の実装 (4-7 週目)

各課題について示した週はクラス全体に向けての説明を行う週の目安であり、実験の進行状況によっては前後することもある。本レジュメには全ての課題について記載してあるので、余裕があれば先の実験を進めておくこと。

2.2　場所

クリエーションコア 3F 知能情報実験室 1(CC301) にて実験を行う。一人 1 台の PC を用いて各自でシミュレータ及びロボット向けのプログラムを作成する。

2.3　レポート

タイトル、氏名、学籍番号、日付をいれた表紙をつける。本レジュメに記載された各演習課題について、

1. 演習課題の内容説明
2. アルゴリズムの説明 (できるだけ詳しく)
3. 動作結果（どういう挙動を示したかを詳しく）
4. 考察（その挙動の理由の分析）
5. プログラムリスト

を掲載してレポートを作成する。授業中に指定する日時 (最終授業日の 2 週間程後を予定) までにレポート (PDF) 及び実際に使用したプログラムファイル (C++) を **manaba+R** に提出すること。

2.4 サポートウェブサイト

本テーマのサポートウェブサイトは次の URL からアクセスできる。

`http://www.i.ci.ritsumei.ac.jp/darwin/`

知能情報学科のホームページからも辿ることができる。このサイトにこのレジュメファイル、サンプルファイル、その他必要な情報などを掲載するので確認しておくこと。

3 ヒューマノイド・ロボット DARwIn-OP

3.1 概要説明

本テーマでは、ROBOTIS 社製の小型ヒューマノイド・ロボット DARwIn-OP (Dynamic Anthropomorphic Robot with Intelligence–Open Platform) [1] (図 1) を用いる。このロボットは自身がコンピュータを搭載しており、Linux OS で制御される。キーボードやマウス、モニタを繋ぐとそのままパソコンとして利用することもできるため、プログラミングや制御に柔軟性がある。首、腕、足などの稼働部分は、それ自体が通信能力をもったインテリジェントモーター 20 個を接続して構成されている。カメラ、ジャイロ、加速度センサ、無線・有線 LAN も搭載している。バッテリ駆動も可能（現時点では稼働時間が短いので授業では使わない)。詳しく知りたい場合は ROBOTIS 社のオンラインマニュアル[2] を参照のこと。

ジャイロや加速度センサを利用した非常に高い適応的な歩行能力を標準で搭載している。もし倒れても倒れたことを検知して自分で起き上がるようにプログラムできる。

[1] `http://jp.robotis.com/index/product.php?cate_code=111010`
[2] `http://support.robotis.com/en/product/darwin-op.htm`

図 1: DARwIn-OP

3.2 起動と停止の仕方

DARwIn-OP はパソコンと同様のコンピュータなので、起動と停止はきちんと次の手順を守って行うこと。

起動

1. 図 2 に示す安全姿勢をロボットに取らせる。崩れ落ちないようにしっかりと据え置くこと。

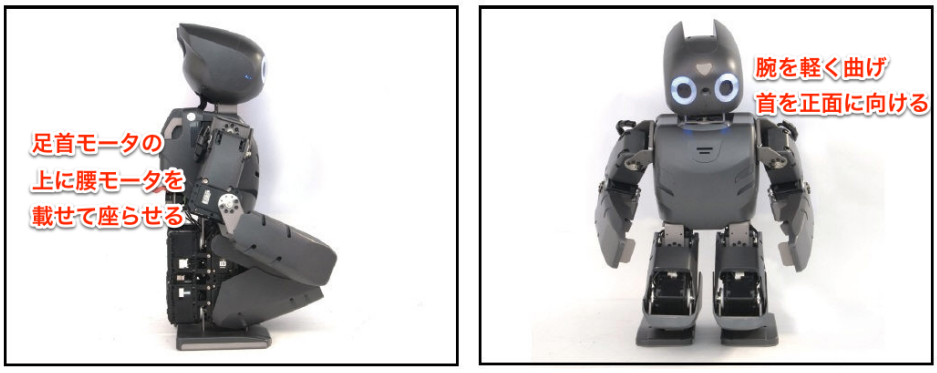


図 2: 起動時の安全姿勢

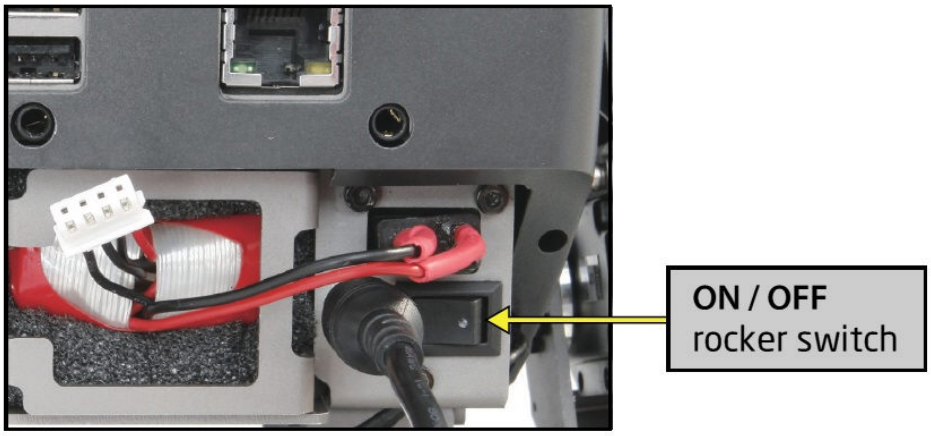


図 3: AC アダプタの接続と電源スイッチ

2. AC アダプタをしっかりと差し込む（図 3)。
3. 背面の電源スイッチを入れる（図 3)。
4. “Demonstration ready mode” と声がして額の LED が黄色に光って正常起動したことを確認する。
5. Webots シミュレータから制御するには、起動している demo プログラムを停止する（ 5.5 節参照)。額の LED が緑色になる。

停止

DARwIn-OP を停止させる際は、以下の手順に従う。

1. DARwIn-OP の動作が停止していることを確認する。動いているときは、背面の吊り手をしっかりと持ってから、背面のリセットスイッチ (図 30) を押して動作を止める。リセットスイッチを押すと全てのモータのテンションが切れ、機体が崩れ落ちるので必ず吊り手で支えること。
2. Webots と実機の接続が切断されていることを確認の上、胸のシャットダウンスイッチ (図 4) を短く 3 回押す。
3. “Bye bye” と声がして額の LED が点滅を開始する。額の LED が消灯したらシャットダウンが完了したので、背面の電源スイッチを切る (図 5)。

4　シミュレータと実機

ロボットのプログラミングは実際に動かしてみないとアルゴリズムが正しく実装されたかどうかわからないことが多いため、予期しない無理な動きが発生すると機械が破損する。これを避けるため、今回の授業では DARwIn-OP を公式にサポートしているロボットシミュレータアプリ Webots[3] (Cyberbotics 社製) を用いる (図 6)。

[3] http://www.cyberbotics.com/overview

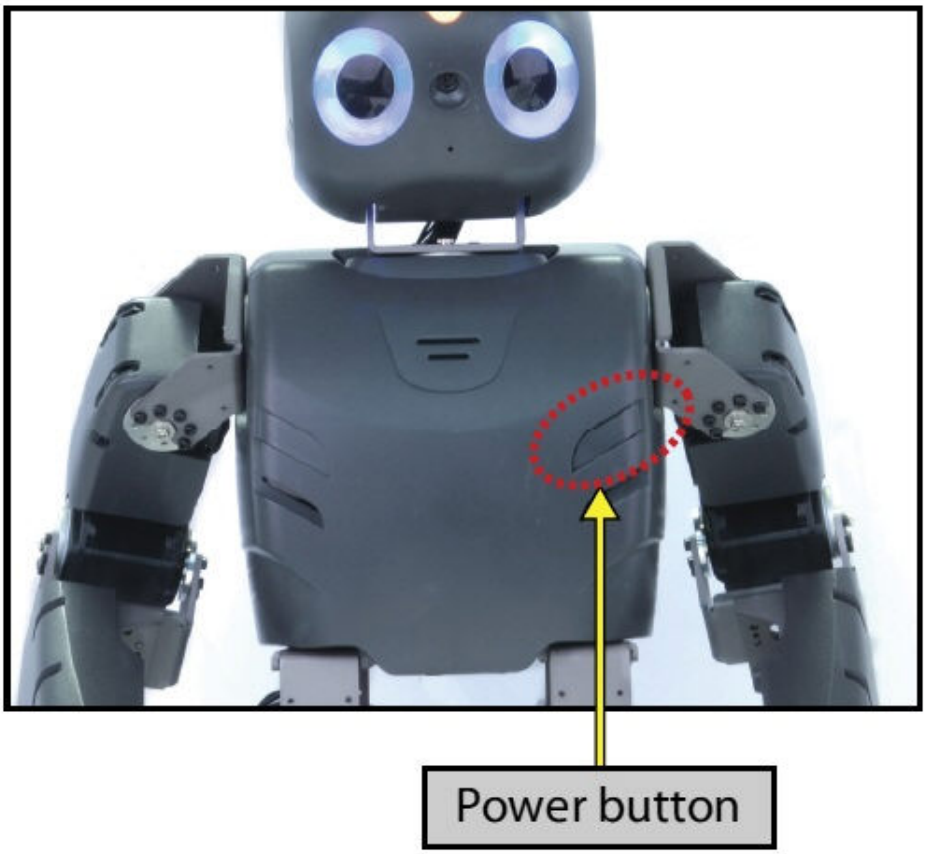


図 4: シャットダウンスイッチ

シミュレータ上の仮想環境には 3 次元 CG による DARwIn-OP が再現されており、物理シミュレーションエンジンによって重力や接触力などを考慮して実環境と同様に動作する。シミュレータ上で制御プログラムを書き、コンパイルして実行すると、仮想環境中のロボットが動作する。この動きを確認しながら、書いたプログラムに問題がないかどうかを検証して、大丈夫そうなら実機に転送して実行する（リモート実行）。ボタンひとつでシミュレータ上と全く同じプログラムをつかって実機ロボットが動作する。一部歩行などの高速フィードバックが必要な制御はプログラムを転送して実機上で動作させることもできる。

プログラミングの使用言語は C++で書かれるがしかしほとんど C 言語だとおもって使えるようにしてある。各課題ごとにサンプルプログラムコードをいくつか提示するので、それを改造する形で課題をやり、最後にそれらを組み合わせてロボットのサッカープログラムをつくってみる（しかし意外とむずかしい！）。

シミュレータアプリを同梱した Linux の Live DVD を各受講生に配布する。授業では学科の実験室 PC で起動して演習するが、他の PC のドライブにいれると Linux を起動して実行することもできるようになっている。インターネット認証のライセンスアカウントがすでに埋め込まれているので、インターネットとの通信が可能な環境であればどこでも好きなだけシミュレータを使って自学自習できる。

シミュレータでのプログラミング課題はすべて個人で行う。レポート提出も個人単位とする。実機は大体 2 人から 3 人に 1 台割り当たるので、うまくシェアしてシミュレータで確認したのち実機で試しながら課題を行うこと。

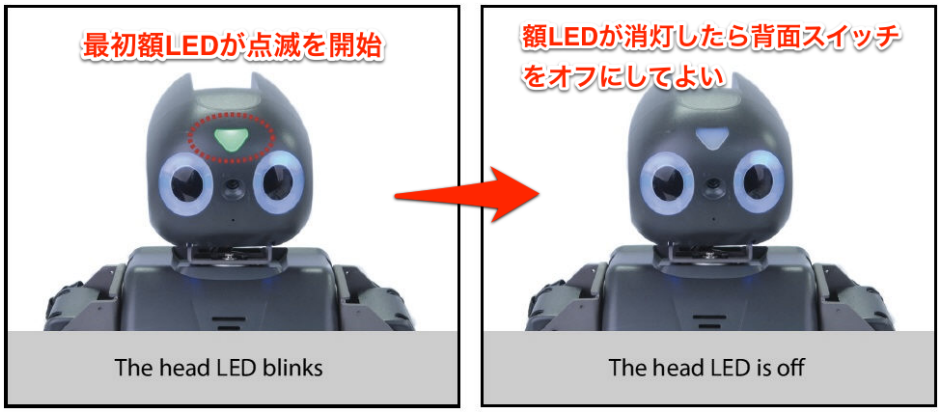


図 5: シャットダウン時の LED の変化

5 Webots の起動手順

本テーマでのロボットの制御プログラムの開発と動作テストは Webots というシミュレータソフトウェア上で行う。Webots 導入済みの Live DVD を用意してあるので、これを用いて Webots を起動する。

この Live DVD はデスクトップ PC、ノート PC のどちらの環境にも対応しているが、OS(Linux) 起動までの手順が一部異なる。それぞれの手順と、OS 起動後に Webots を起動する方法及び Webots のライセンス管理システムについて以下に述べる。

5.1 VMware player での Live DVD 起動

Live DVD をドライブに挿入した状態で、物理ドライブを起動デバイスとして設定した仮想マシン (VM) を起動すればよい。

具体的な手順は以下の通り。

1. PC の電源を入れ、Windows にログオンする。
2. DVD ドライブに Live DVD を挿入する。
3. 本テーマのサポートウェブサイト (`http://www.i.ci.ritsumei.ac.jp/darwin/`) にアクセスし、仮想マシン設定ファイル (`dvd-vmx.zip`) をダウンロードする。
4. ダウンロードした ZIP ファイルを展開し `webots.vmx` をダブルクリックする。拡張子の異なる同名のファイルが複数あるので注意する。ダブルクリックするファイルのアイコンは図 7 を参照すること。

実験室では VMware Player による仮想マシンを用いるが、物理デバイスを起動デバイスにできる仮想マシンであれば Live DVD での環境を起動できる。

5.2 デスクトップ PC での Live DVD 起動

デスクトップ PC の場合、Live DVD をドライブに挿入した状態で電源を入れれば Linux が起動できる。DVD を挿入する前に WindowsXP, Vine Linux 等のネットワーク起動のための OS 選択画面になってしまった場合は、選択を行わず DVD 挿入後にリセットボタンを押せば Live DVD が起動できる。

1. Click on the Stop button
2. Click on the Revert Simulation button.

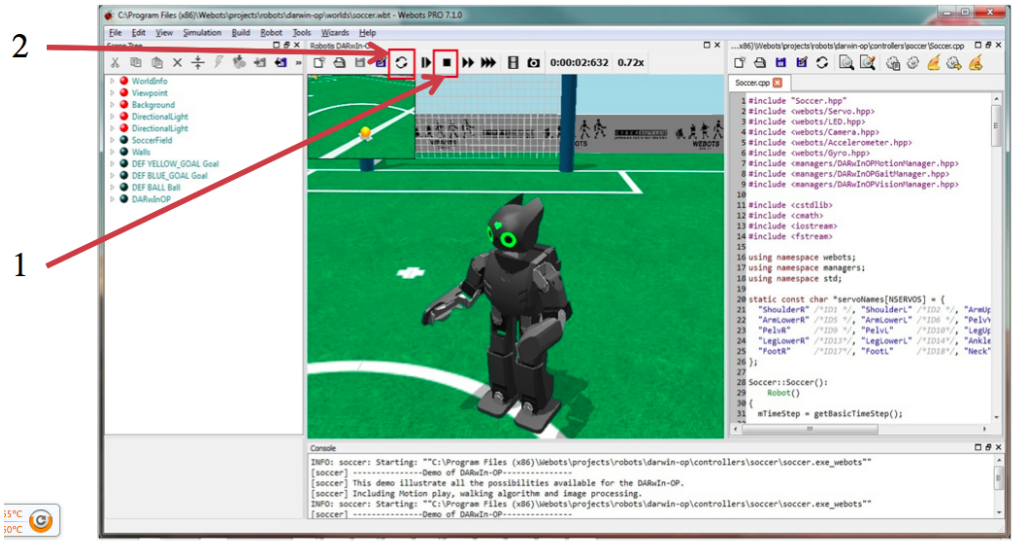


図 6: Webots

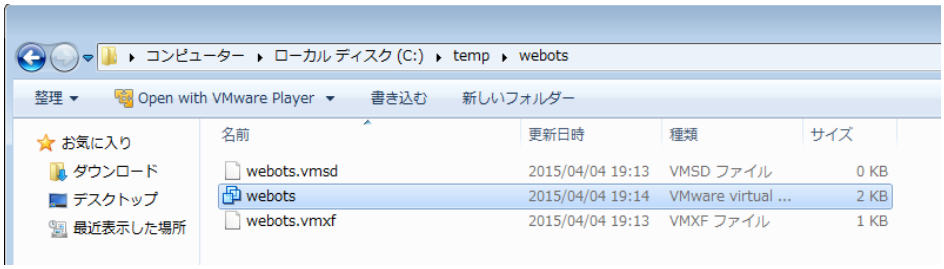


図 7: VMware Player 用仮想マシン設定ファイル

5.3 ノート PC での Live DVD 起動

ノート PC の場合、ドライブに DVD を挿入するだけでは Live DVD が起動されない (UEFI によるセキュリティ確保のため)。Live DVD を起動するための手順は以下の通り。

1. 電源スイッチを入れる。
2. 電源投入後、10 秒以内に `ESC` を押す。DVD ドライブに Live DVD を挿入する。
3. 図 8 のようなメニューが表示されるので、`F9` を押す。
4. 図 9 のようなメニューが表示される。上下の矢印キーでカーソルを動かして `Internal CD/DVD ROM Drive` を選択して `Enter` を押す (図 10)。

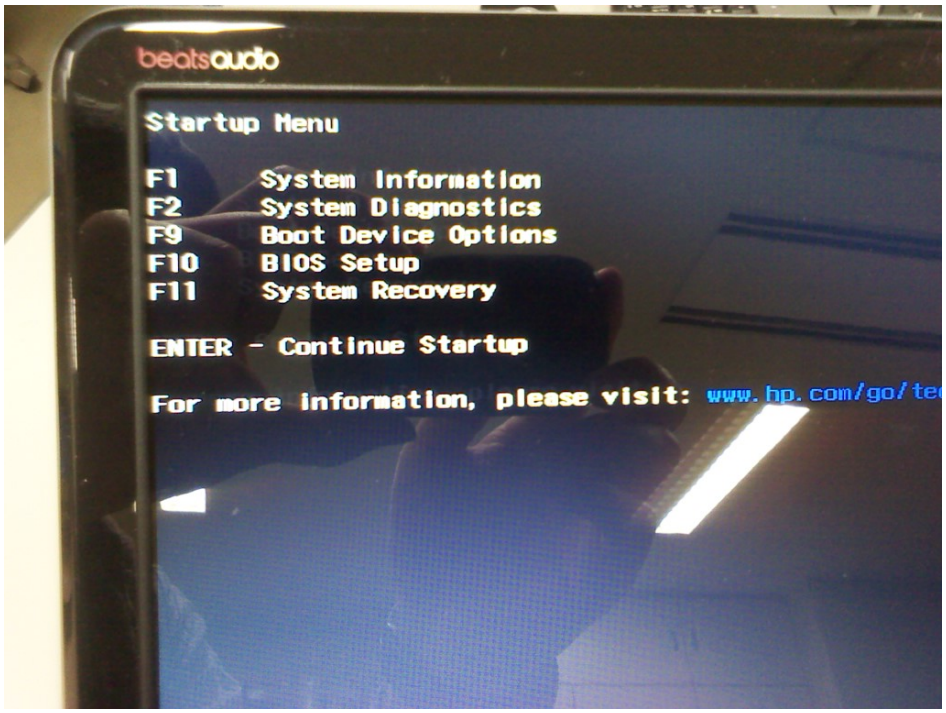


図 8: Startup Menu

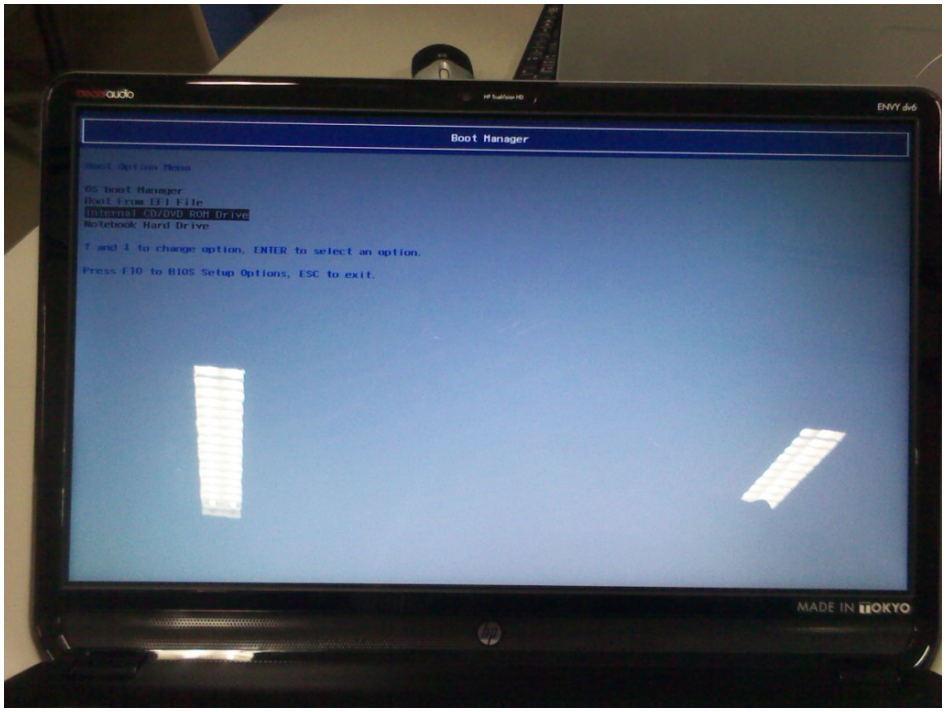


図 9: Boot Manager

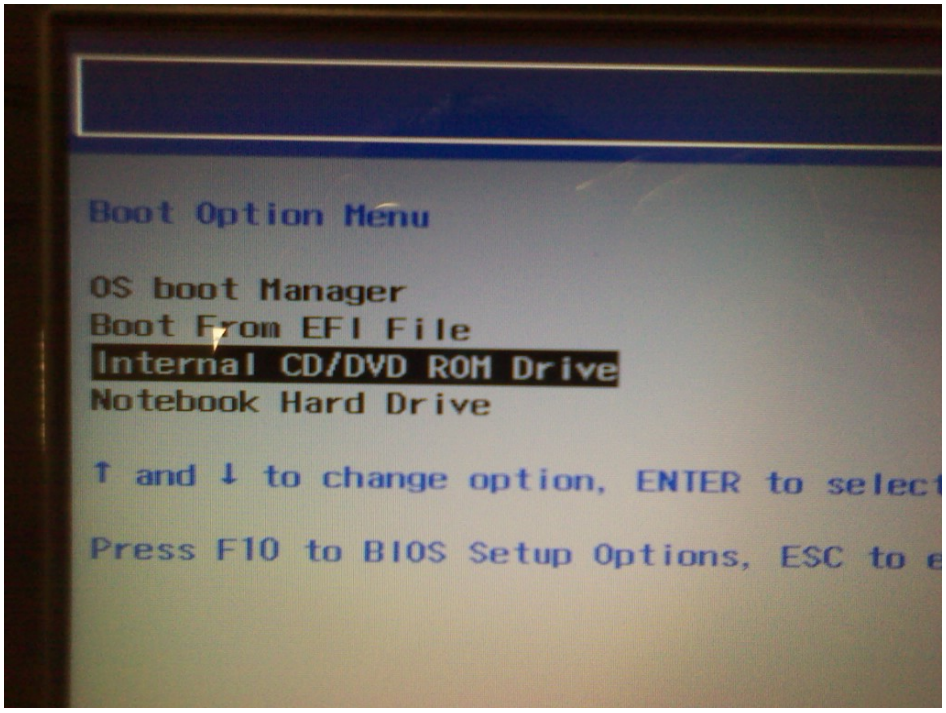


図 10: Boot Manager で選択する項目

5.4 Webots の起動

Live DVD が起動すると、図 11 のようなメニューが表示されるので `Live()` を選択して `Enter` を押す。

しばらくすると Live DVD の図 12 のようにデスクトップが表示された状態となる。
デスクトップ上の `Webots` のアイコン (図 13) をダブルクリックすれば Webots を起動できる。

5.5 DARwIn-OP 上の Demo プログラムの停止

新しく作成したプログラムをロボット DARwIn-OP の実機上に送り込む際、DARwIn-OP の電源投入時に自動的に実行されるプログラム `demo` は自動的に終了される設定となっている。

新しいプログラムを送り込まずに DARwIn-OP 上で動いている `demo` などのプログラムを停止させるにはデスクトップ上にある `Kill demo` のアイコン (図 14) をダブルクリックすればよい。このアイコンをダブルクリックして実行すると、対象となる DARwIn-OP の番号を尋ねられる (図 15) ので、2 桁の番号で入力して `Enter` を押す。`D01` の場合は `01` を、`D25` の場合は `25` を入力する。続いてパスワードを尋ねられるのでここでは `111111` を入力して `Enter` を押す。以上の手続きで DARwIn-OP 上のプログラム `demo` が終了される。`demo` が終了すれば DARwIn-OP の額の LED が緑色となる。

5.6 Live DVD 環境のシャットダウン

Live DVD 環境を終了する際にはデスクトップ左上にあるログアウト用のアイコン (図 16(a)) をクリックする。ログアウトダイアログ (図 16(b)) が開くので、ここで**電源を切る**をクリックすると Live DVD

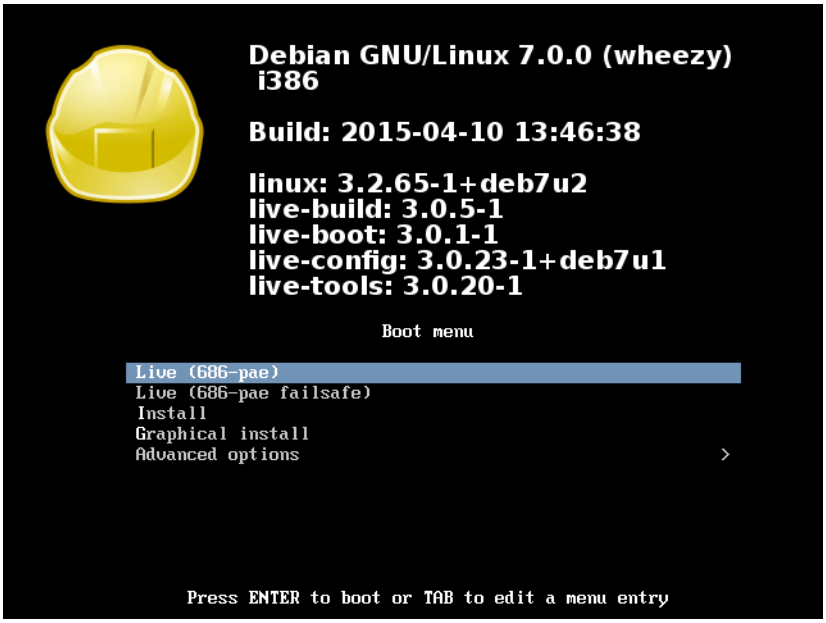


図 11: Live DVD の boot menu

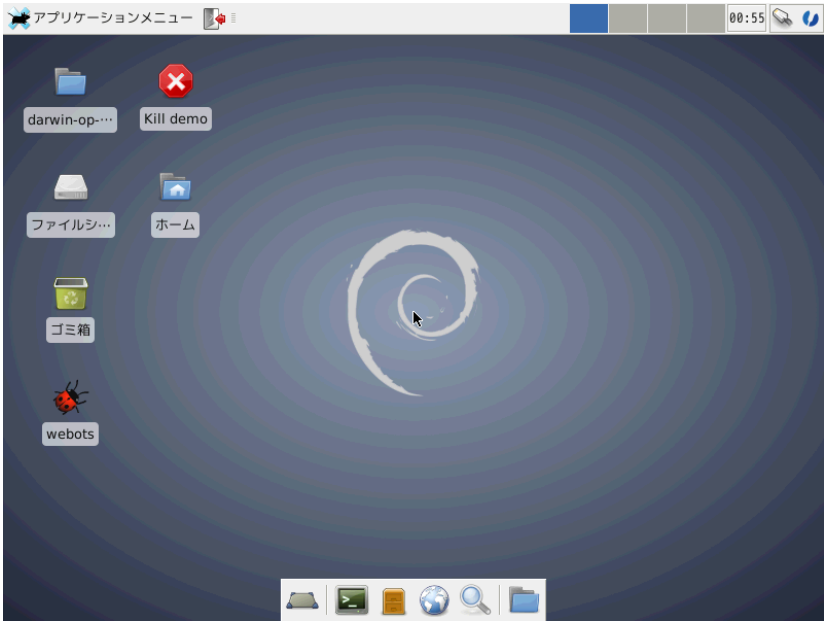


図 12: Live DVD のデスクトップ

図 13: Webots のアイコン

図 14: DARwIn-OP 上のプログラム停止用アイコン

図 15: demo 停止プログラム

(a) ログアウト用のアイコン

(b)「電源を切る」でシャットダウンできる

図 16: Live DVD 環境のシャットダウン

環境をシャットダウンすることができる。

その際、*Live DVD* の環境内に保存されたファイルは全て失われることに注意する。必要なファイルは USB メモリやオンラインストレージ等に保存しておくこと。

5.7 Webots のライセンス管理

Webots を使用するにはインターネット経由のライセンス認証が必要となる。Live DVD 上の Webots はライセンス情報が既に設定済みであるが、その認証のためにインターネットへのアクセスが必要となる。実験室以外の環境で Live DVD を使う場合には注意すること。

6 課題０：サンプルプログラムの読み込みと実行

この課題は練習なのでレポートには含めない。

6.1 wbt ファイルの読み込み

Webots ではシミュレータ内の環境は wbt ファイルに保存される。あらかじめいくつかの wbt ファイルを用意してあるので、適宜使い分けること。

課題 0 用の wbt ファイルを開く手順は以下の通りである。

1. 本テーマのサポートウェブサイト (`http://www.i.ci.ritsumei.ac.jp/darwin/`) にアクセスし、サンプルプログラムキット (`darwinop-sousei.zip`) をダウンロードする (図 17)。
2. ダウンロードマネージャ上の ZIP ファイルを右クリックし、保存先のフォルダを開く (図 18, 19)。
3. ZIP ファイルを右クリックし、メニューからここで展開を選択する (図 20)。
4. `darwinop-sousei` というフォルダが作られる (図 21) ので、このフォルダを右クリックし、コピーを選択する (図 22)。
5. デスクトップのホームのフォルダを開き、余白の部分で右クリックして貼り付けを選択する (図 23)。

   他のフォルダに貼り付けて利用することもできるが、フォルダのパス文字列に全角文字が含まれていると *Webots* が正しく動作しないので、パスが全て ASCII 文字となるよう注意すること (「創成」などのフォルダ名を使わない)。
6. ホームフォルダである`/home/user` 上に `darwinop-sousei` というフォルダが生成される。
7. Webots を起動し、メニューの `File` を開いて `Open World...` を選択する (図 24)。
8. 表示されたダイアログでホームフォルダ`/home/user` 上の `darwinop-sousei/worlds/field.wbt` というファイルを開く (図 25(a),(b))。
9. `field.wbt` の環境が開く (図 26)。
10. この環境でプログラムの追加、変更等を行う。

worlds フォルダには wbt ファイルがあり、それぞれ下記のようなシミュレーション環境が記録されている (図 25(b))。

図 17: ZIP ファイルを保存する

図 18: ZIP ファイルの保存先を開く

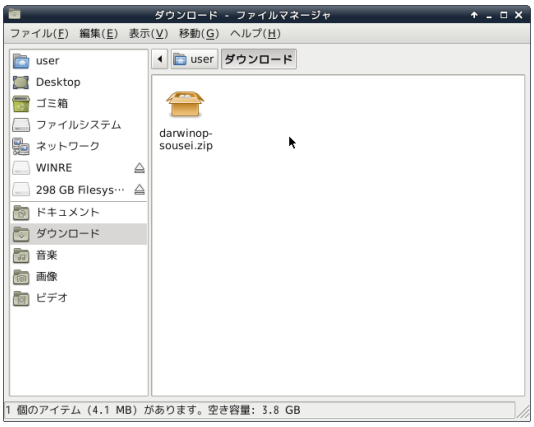


図 19: ZIP ファイルの保存先

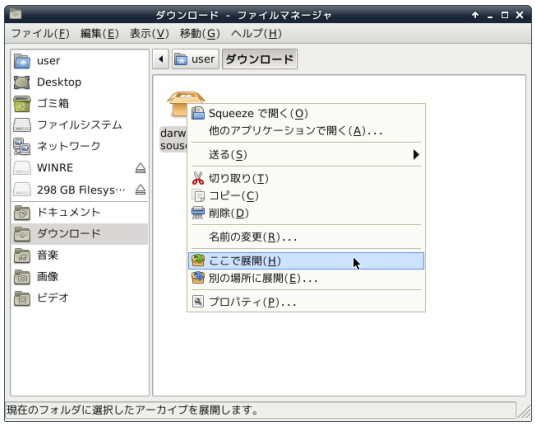


図 20: このフォルダで展開する

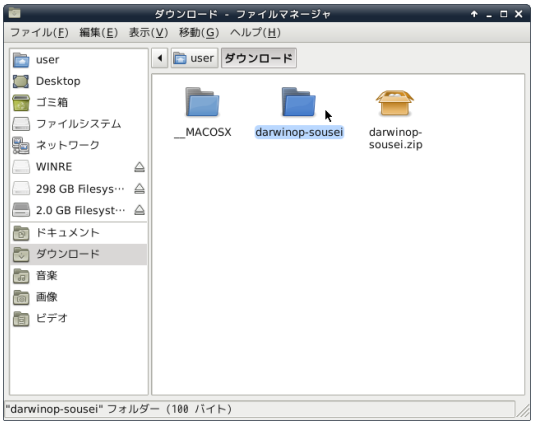


図 21: darwinop-sousei を選択する

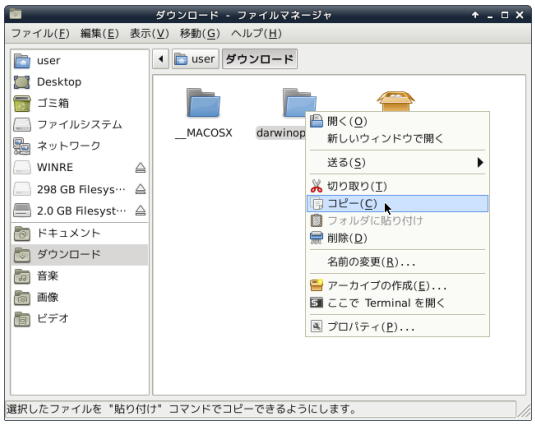


図 22: フォルダをコピーする

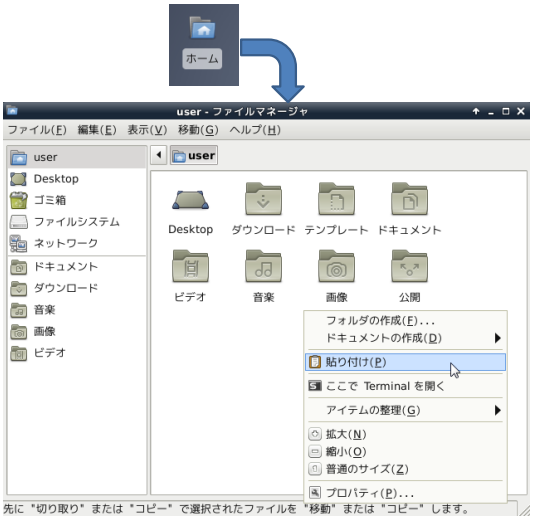


図 23: ホームフォルダに貼り付ける

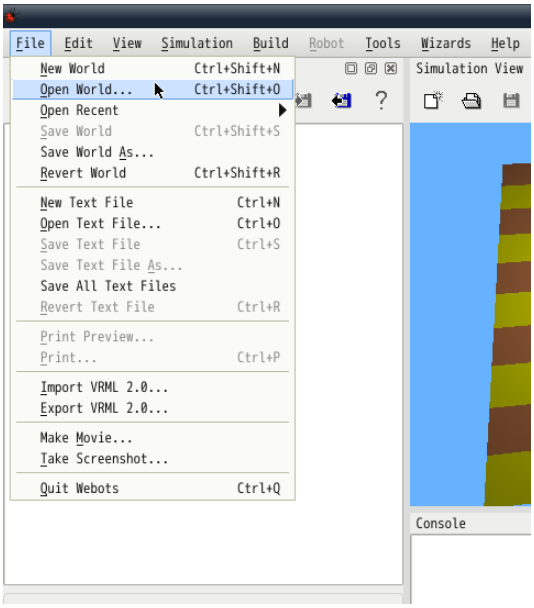


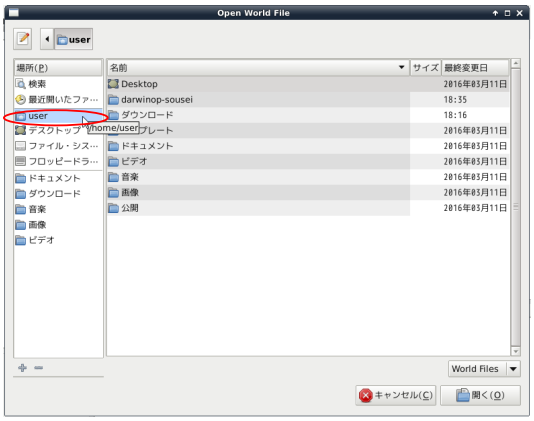
図 24: Webots の Open World... を選択する

field.wbt 課題 0〜4 のプログラムを実行するための環境。

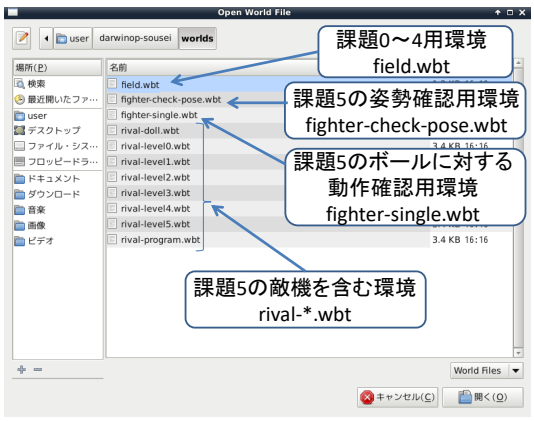
fighter-single.wbt 課題 5 でロボットのボールに対する動作を確認するための環境。武器とボール台を装着したロボット (自機) に加え浮遊するボールが含まれている。

rival-doll.wbt 課題 5 で自機ロボットが敵機ロボットに接触した場合の動作を確認するための環境。武器とボール台を装着したロボットが 2 台 (自機と敵機) が含まれている。敵機は静止しており、その姿勢は `doll.cpp` を書き換えることで変更できる。

fighter-check-pose.wbt 課題 5 のプログラム作成時に、関節角度と姿勢の関係を確認するための環境。



(a) 左ペインでホームフォルダ `/home/user` を選択する



(b)`worlds` フォルダ内の `wbt` ファイルを選択する

図 25: `field.wbt` を開く

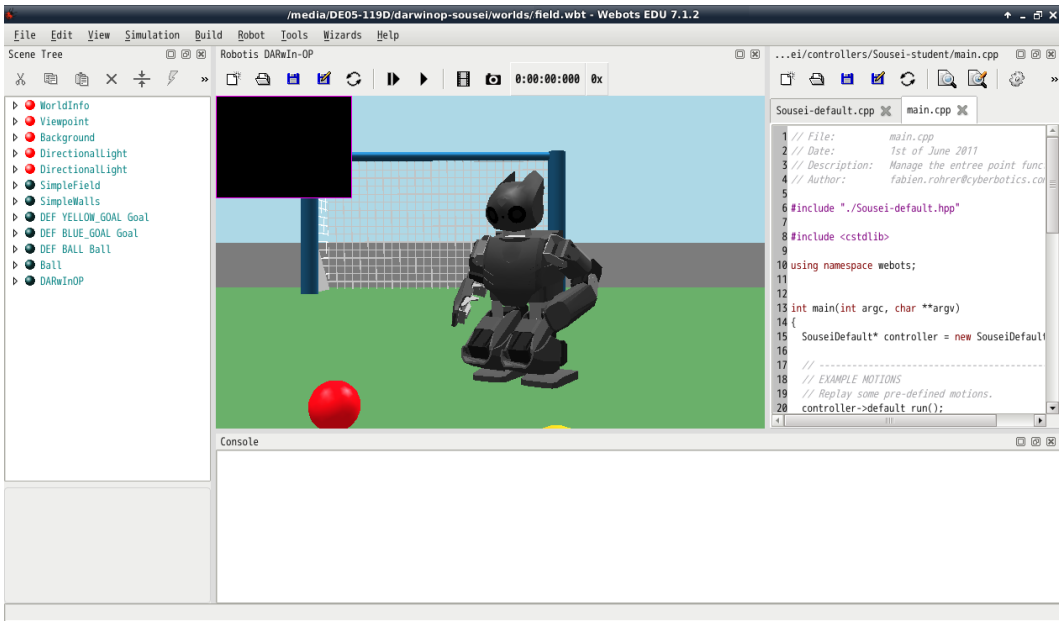


図 26: `field.wbt` の環境

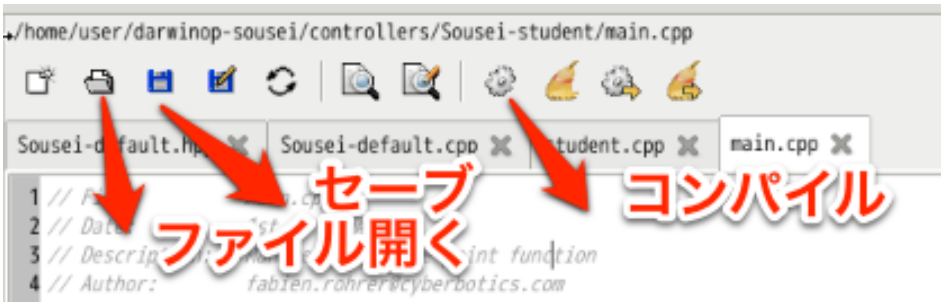


図 27: Webots のエディタ画面操作ボタン

6.2 コントローラーのコンパイルと実行

wbt ファイルを読み込むと、中央の環境 CG パネルに DARwIn-OP の仮想モデルが表示され、右側のエディタ画面にはコントローラーのプログラムソースが表示される。サンプルプログラムは `main.cpp`, `soccer.cpp`, `Sousei-default.hpp`, `Sousei-default.cpp` の 4 つのファイルで構成されている。

エディタ上部の歯車マークを押すとコンパイルが実行される (図 27)。コンパイルされる様子はウィンドウ下部の出力パネルにメッセージとして出力される。もしコンパイルエラーがあれば赤字で出力される。

コンパイルが無事完了したら、環境パネルの上部にあるシミュレーション再生ボタンを押すと、シミュレーションがスタートする (図 28)。サンプルプログラムの最初の状態では、ロボットにプリセットされた行動 (拍手 (Hello) やキック (LeftKick,RightKick)) が順番に実行される。 `main.cpp` を開くと、`default_run()` という関数が呼び出されている。この関数は `Sousei-default.cpp` に定義されている。`playPage(LeftKick)` という関数は左足によるキックをする、という命令である。どんなプリセット行動があるかは `Sousei-default.hpp` に定義されているので確認してほしい。

図 28: シミュレーション操作ボタン

6.3 仮想環境内での視点と物体の操作

環境パネル内でのマウス操作で、仮想環境を表示する視点の変更や、仮想環境内の物体の移動・回転などが行える。シミュレーション実行中にも行えるので、状況の変化に対してロボットが期待通りに動作するかの検証を行うことができる。

視点を変更する方法は以下の通りである。

天球上でカメラを移動させる
: 左ボタンを押したままマウスを移動 (左ドラッグ) する。

スクリーンに対して平行にカメラを移動させる
: 右ボタンを押したままマウスを移動 (右ドラッグ) する。

スクリーンに対して前後にカメラを移動させる
: 真ん中ボタン (ホイール) を押したままマウスを移動 (真ん中ドラッグ、ホイールドラッグ) する。タッチパッドの場合は左右ボタンを両方押せば真ん中ボタンを押した扱いとなる。

物体を移動・回転させる方法は以下の通りである。

物体を水平に移動させる
: 環境パネル内で物体をクリックして選択状態にし、物体上で左ボタンを押したままマウスを移動 (左ドラッグ) する。ドラッグ状態でマウスを上下左右に動かすと物体が仮想環境内の水平面内で移動する。

物体を鉛直方向に移動させる
: 環境パネル内で物体をクリックして選択状態にし、物体上で真ん中ボタン (ホイール) を押したままマウスを移動 (真ん中ドラッグ、ホイールドラッグ) する。ドラッグ状態でマウスを上下に動かすと物体が仮想環境内の鉛直方向に移動する。

物体を回転させる
: 環境パネル内で物体をクリックして選択状態にし、物体上で右ボタンを押したまま `Shift` キーを押す。`Shift` キーを押すたびに表示される回転軸が切り替わるので、回転させたい軸が表示された状態で `Shift` キーと右ボタンを押したままマウスを移動させれば物体を回転できる。

上記は仮想環境内の物体の位置と姿勢のパラメータを変更して物体を移動・回転させるもので、移動量・回転量は物体の大きさ、重さなどに依存しない。

これとは別に、仮想環境内に物体に作用する力を発生させることもできる。

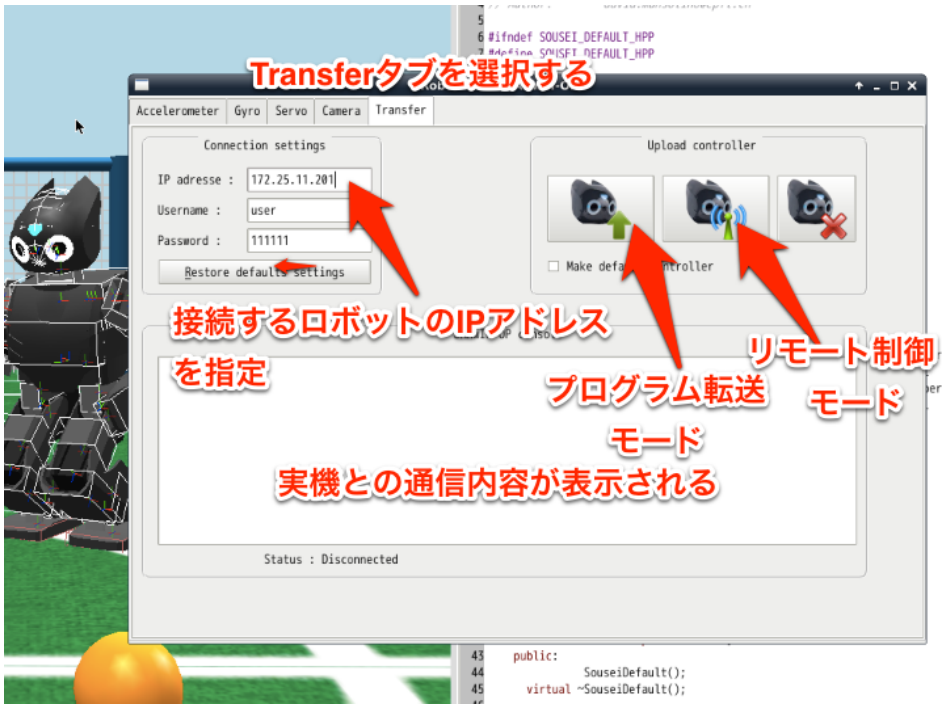


図 29: ロボットウィンドウ

物体に力を作用させる

シミュレーション実行時に作用させたい部分を数回クリックして選択状態にする。選択した部分上で `Ctrl` キーと `Alt` キーと左ボタンを押したままマウスを移動させると、選択した部分からマウスカーソルまでを結ぶ矢印が表示される。この矢印が表示されている間は、矢印の方向の力が選択した部分に作用する。また、矢印の長さが力の大きさに対応しており矢印が長いほど強い力となる。

ロボットのように多数の部品からなる物体の一部分を選択するには、仮想環境内でその部分を数回クリックすればよい。

図 28 の時刻巻き戻しボタンをクリックすると、上記の視点の変更や物体の移動・回転などの結果も全てリセットされ、wbt ファイルを読み込んだときの状況に戻る。メニューの `File` から `Save World` を実行すると現在の状況が wbt ファイルに保存される。DARwIn-OP も移動・回転可能であるが、無闇に動かすとシミュレーション開始時に安定姿勢を取れない恐れがあるので注意すること。

6.4 シミュレータから実機をリモート制御するモード

シミュレータ上での動作に異常（急激な動きや関節の過剰な回転、転倒など）がなければ、実機を動作させる事ができる。

シミュレータ上のコントローラープログラムを実機に接続するには、環境パネル中の DARwIn-OP をマウスでダブルクリックする。するとロボットウィンドウが開く（図 29）。一番右の transfer タブを開くと、左に通信したいロボットの IP アドレスと接続時のユーザ名・パスワードを入力するパネル、右側にはボタンが 3 つならんでいる。IP アドレスは “172.25.11.2**”（**はロボットの胸についている二

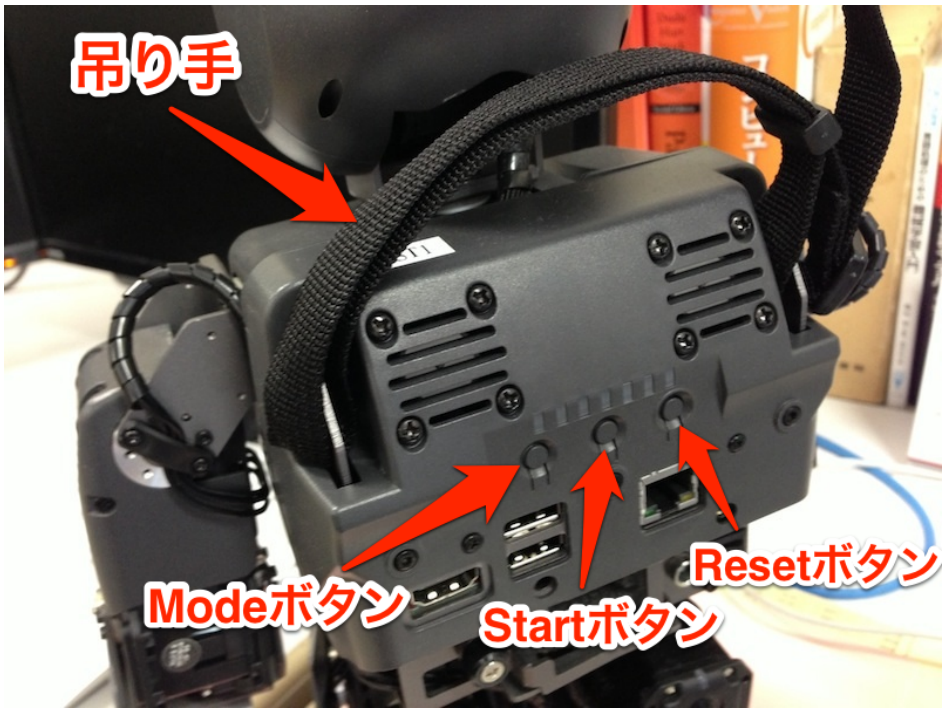


図 30: DARwIn-OP 背面図

桁の番号) を入力する。ユーザ名とパスワードはそれぞれデフォルトの “darwin”、“111111” のままでよい。

入力し終わったら、右の３つのボタンのうち真ん中のボタンを押すと通信がはじまる。通信回線が確立したメッセージが出たら、もとのウィンドウの環境パネルに戻り、シミュレーション再生ボタンを押すと CG モデルの代わりに実機が動き始めるので、確認する。停止するときはシミュレーション再生ボタン（停止ボタンになっている）をおして動作を止め、さきほどのロボットウィンドウで押した真ん中のボタンを再度押して通信回線を切断する。

（注意）ロボットは転倒する可能性があるので机の上では動作させず（落下すると危険)、床面に設置すること。また背中の保持紐を軽く持った状態で転倒にそなえること。歩くときは電源のコードを持ってロボットが踏まないように気をつけること。

（注意２）ロボットの動作が止まらない、モーターの無理な回転が続く場合は、即座に背中の釣り手を掴んでロボットを持ち上げ、背中のリセットボタン（３つあるボタンのうち一番右のボタン）（図 30）を押す。このとき関節に指などを挟まないように細心の注意を払うこと。電源スイッチを切ると完全停止するが、OS が強制終了するのでロボットの内部ファイルが破損する可能性があるため、極力避けること。

6.5　実機へプログラム転送して実行するモード

前節の方法では、コントローラのプログラムはシミュレータ上で動作しており、ネットワーク通信経由で実機を制御している。このため制御のための通信に時間がかかり（オーバーヘッドが大きい、という）歩行などの高速な動作をさせると正常に動作しない。

そのような場合には本節で述べるプログラムの実機転送実行モードを利用する。この方法ではシミュレータ上でつくったコントローラーのプログラムを実機に転送し、実機上ですべての動作を実行するため、余計なオーバーヘッドがなく高速に動作する。

操作方法は、前節とほぼ同じで、ロボットウィンドウの一番左のボタンを押す。すると転送がはじまり、転送が終了したら自動的に実機の実行が始まる。実行させると実機は自律的に動作をする。停止させるにはおなじ一番左ボタンを再度押すと動作が停止し、通信回線が切断される。

6.6 キーボードによる歩行のリモコン制御

main.cpp を変更して、コントローラーの動作をリモコン歩行に変える。最初は default_run() を呼び出しているのでこれをコメントアウトし (//を行の先頭につける)、walk_run() の先頭のコメントマークを削除してこの行を生かす。そしてコンパイルして、シミュレータ再生ボタンを押す（実機では動作させないこと)。

キーボードの操作でロボットが動作するので、試してみること。

1. スペースバー：歩行の開始・停止
2. 上矢印：前進　下矢印：後退
3. 右矢印：右旋回　左矢印：左旋回
4. A：左サイドステップ　S：右サイドステップ
5. K：右キック J：左キック

キーボード操作は現在の Webots シミュレータでは、実機動作できない（制御が遅く間に合わない）のでシミュレータのみで実行する。

7 課題１：マススレーブの実装

マススレーブとは、手元のロボットを動かすと、同じ姿勢にあるように別のロボットが動作するようなしかけのことである。この課題では、ロボットの右腕をマスター側（操作側)、左腕をスレーブ側（被操作側）としてマススレーブを作ってみる。

7.1 サンプルプログラムの実行

サンプルプログラムは symmetry_run() である。main.cpp を前の課題と同様に変更してコンパイルし、実行してみる。

7.2 シミュレータ上のロボットの操作

シミュレータでは、環境パネル上のロボットの各部をつまんで力をかけることができる。ここでは右腕を動かしてみる。

1. マウスでロボットの右下腕部分をクリックする。

2. ロボットの全身ではなく右下腕だけが白いワイヤーフレームで強調表示されるようになったら、その状態でキーボードの Ctrl キーと Alt キーを同時に押しながらマウスの左ボタンをドラッグする。
3. 腕からマウスの先端に赤い矢印が伸びるようになる。矢印が長いほど強い力で引っ張っていることになる (6.3 節参照)。
4. マウスを動かしながら引っ張る方向と強さを調整すると右腕が動くので確認する。
5. 右腕が動くと左の肩関節がいっしょに動くのを確認する。

Sousei-default.cpp の symmetry_run() 関数の構成は以下のようになっている。

1. setMotorForce() 関数で右腕の力を 0 にする（自由に動かせるようになる）
2. while ループの中で getPosition() 関数をつかって右腕の各関節モーターの現在の角度値を pos0 に格納している。
3. pos0 の値を setPosition() 関数を使って符号反転してから左腕の関節角度に代入している。

右腕と左腕は対称なので、各関節の角度の符号を反転させると左右対称な動きを実現できる。

【演習課題 1 】

symmetry_run() 関数はいくつかの行を意図的にコメントアウトしている。これを復元し、さらにプログラムコードを付け加えて、腕のすべての関節 3 つが左右対称な動きをするように改良せよ。また首の左右の回転角度が右肩の回転角度と連動するようにプログラムコードを追加せよ。

シミュレータで正常な動作確認したら、課題 0 の要領で実機と接続して動作させてみよ。

（ヒント：使える関数の説明、定義済み変数、定数の説明はこのレジュメの末尾にある）

ソースコード 1: symmetry_run()

```
//-------------------------
//
// symmetry_run()
//
//-------------------------
void SouseiDefault::symmetry_run() {
  cout << "-------Symmetry example of DARwIn-OP-------" << endl;
  cout << "The right arm is free while the left one mimic it." << endl;
  cout << "In order to move the left arm, select the robot with the mouse," <<
       endl;
  cout << "press ctr+alt and select the right arm." << endl;
  cout << "Now you just have to move the mouse without releasing it." << endl;
  cout << "This example illustrate also the selfCollision which is activated
      by default" << endl;

  // LED
  setHeadLEDColor(0x00FF00);

  // free right arm
```

```
  setMotorForce(ShoulderR,0.0);
  setMotorForce(ArmUpperR,0.0);
  setMotorForce(ArmLowerR,0.0);

  // step
  myStep();

  while (true) {

    // Get position of right arm of the robot
    double pos0 = getPosition(ShoulderR);
    //double pos1 = getPosition(ArmUpperR);
    //double pos2 = getPosition(ArmLowerR);

    // Set position of XX arm of the robot
    // the inversion of sign is done because of the symmetry
    setPosition(ShoulderL,-pos0);

    // step
    myStep();
  }
}
```

8　課題２：首ふり（２自由度）の制御プログラムの実装

首関節の制御を学ぶ。首はNeck（横旋回:pan）とHead(縦旋回:tilt)の２つの自由度がある。headmotion_run()関数は、Headの縦旋回をなめらかに行う動きを実装してある。

【演習課題２】

headmotion_run()関数を変更して、首の横旋回(Neck)も行うように改良せよ。このとき首がおおよそ円（楕円）運動するようにするにはどうしたらよいか。考えて実装してみよ。

シミュレータで正常な動作を確認したら、実機で動作させてその様子を観察せよ。

ソースコード 2: headmotion_run()

```
void SouseiDefault::headmotion_run() {
  cout << "-------------------------------" << endl;
  cout << "Automatically Head tilt changes " << endl;
  cout << "-------------------------------" << endl;

  // First step to update sensors values
  myStep();
  setHeadLEDColor(0x00FF00); // cyan head
  setEyeLEDColor(0xFFFFFF); // white eyes

  int i=0;
  while (true) {
```

```
    // write your codes here
    double tilt = 30./180.*M_PI*sin((double)(i)/180.0*M_PI)+20./180.*M_PI;
    setPosition(Head, tilt);

    i++;

    // step
    // Don't delete!!
    myStep();
  }
}
```

9 課題３：画像センサをつかったカラーボールの検出と LED による状態表示

DARwIn-OP には頭部にカメラが搭載されており画像センサとして利用することができる。ここでは色を使ったカラーボールやカラーカードの検出機能（すでに実装してあるもの利用できる）を用いて、検出結果を目の LED カラーをつかって表示する。

9.1 画像センサによるカラー検出

`getRedBallCenter()` 関数は画像中に赤い領域があるかどうかをチェックして、もし見つかったら引数の `x, y` に赤い領域の重心位置をセットして、true の返り値を返す。そうでなければ false の返り値を返す。今回は `x, y` の値は捨てて（しかし引数には指定しないとコンパイルエラーになる）見つかったかどうかだけを利用する。

赤だけでなく、青を検出する `getBlueBallCenter()`、黄を検出する `getYellowBallCenter()` もある。他の中間色も検出させることができるが、今回は使用しない。どうしても使用したい場合は申し出ること。

9.2 LED のカラー指定

DARwIn-OP は両目の周りと額にそれぞれ 3 つの LED を持っている。これらの LED は 24bit で光るカラーを指定することができる。`setEyeLEDColor()` 関数は両目の周りの色を、`setHeadLEDColor()` は額の LED の色をそれぞれ指定する。`0xff0000` は `(RGB)=(255,0,0)` を指定し、`0x0080ff` は `(RGB)=(0,128,255)` を意味する。

9.3 サンプルプログラムの実行

`ballsearch_run()` 関数は、赤色のボールを見つけると目の LED を赤く光らせ、見つからないときは消灯（黒い）させる。

シミュレータ上で、赤いボールをつまんで動かしてみて視界の外に出たら目 LED が消灯することを確認せよ。キーボードで `Shift` キーを押しながらボールをマウスで左ドラッグすればボールを横に移動することができる (6.3 節参照)。また実機でも動作を確認せよ。ロボットには赤色のカードか付属の赤色ボールを見せるとよい。

【演習課題３】

ballsearch_run() 関数を改良して、赤色だけが見つかると目の LED を赤に、黄色だけが見つかると目 LED を黄色に、両方が見つかると緑色に、どちらもみつからないときは消灯（黒）となるようにせよ。

シミュレータで正常な動作を確認したら、実機で動作させてその様子を観察せよ。

ソースコード 3: ballsearch_run()

```
void SouseiDefault::ballsearch_run() {
  cout << "--------------------------------" << endl;
  cout << "Detect red ball and light eyes." << endl;
  cout << "--------------------------------" << endl;

  //Camera On
  enableCamera();

  // step
  myStep();

  while (true) {
    double x, y;
    bool rFound = getRedBallCenter(x,y);
    if( rFound == true ) {
        cerr << "red ball found! << (" << x << "," << y <<")" << endl;
    }

    if( rFound )
        setEyeLEDColor(0xff0000);
    else
        setEyeLEDColor(0x000000);

    // step
    myStep();
  }
}
```

10　課題４：ボールを首振りで追跡制御するプログラムの実装

課題 3 で使った getRedBallCenter() 関数は赤色の領域の重心を引数に代入してくれる。この位置が頭部カメラで撮影した画像の中央になるように首の pan と tilt を制御することが考えられる。これをビジュアルフィードバックという。

getRedBallCenter() 関数の引数に red_x,red_y を指定した場合の変数 red_x, red_y の値とカメラスクリーンとの関係は図 31 のようになる。カメラスクリーンは多数の画素 (pixel) が長方形に配列されたものであり、カメラスクリーンの横の画素数と縦の画素数はそれぞれ getImageWidth() 関数、getImageHeight() 関数で取得できる。カメラスクリーンの中心がカメラの真正面方向となっており、課題 4 ではカメラの真正面が常にボールの方を向くように首の pan と tilt を制御する。

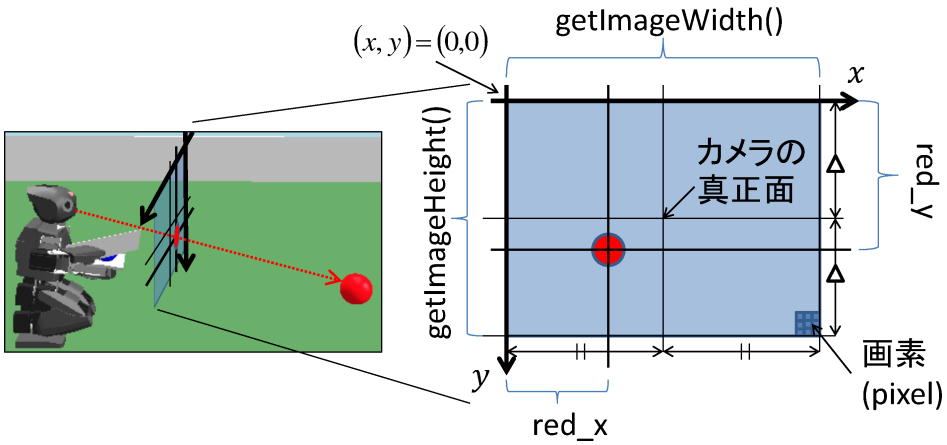


図 31: カメラスクリーン

10.1 サンプルプログラムの実行

`ballsearch_active_run()` 関数は、赤色の領域がカメラ中央になるように、赤色の中心が右によっていたら右に一定角度旋回、左の場合は逆に旋回するアルゴリズムで動作する。縦方向 (tilt) も同様。これをシミュレータで動作させてみよ。ただしこのプログラムを実機で動作させると大変危険なので絶対しないこと。

10.2 追跡制御のアルゴリズム

`ballsearch_active_run()` 関数では、次のようなアルゴリズムで首振りを制御している。

1. 赤色を画像から検知し、見つかったときはその中心座標 (x, y) を取得する。
2. x 座標（横座標）が画像の横幅の `range`(=0.5) 倍より小さいとき、つまり画像の左にあるとき、首 (Neck) を固定角度 (`step_deg`=8 度) 左に回転する。
3. そうでなくて、x 座標が (1-`range`) 倍より大きいとき、つまり画像の右にあるとき、首を固定角度右に回転する。
4. それ以外の場合、つまり画像の中央にあるときは首を動かさない。
5. 同様のやり方で y 座標を元に縦方向（Head) の回転を決める。
6. 以上の動作を画像が取得できるたびに行う（while ループ)。

簡単にいえば、右に見えれば右を向き、左が見えれば左を向く。

【演習課題４】

`ballsearch_active_run()` 関数をシミュレータで実行した結果はどのようなものであったか報告せよ。またこのアルゴリズムのどこに問題がありそうか分析して述べよ。

次にそのような問題が起きないようにこの関数を改良し、きちんと赤色領域が中央になるように首振り動作をするプログラムとせよ。その際どのような考え方にもとづいて、どんなアルゴリズムを考えた

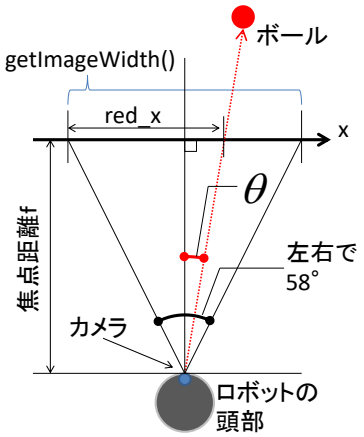


図 32: ボールの方向とスクリーン上の位置との関係

か詳しく説明せよ。途中いくつかの候補アルゴリズムを試した場合はそれぞれやり方を説明し、なにが問題だったかをあわせて説明せよ。

シミュレータで正常な動作を確認したら、実機で動作させてその様子を観察せよ。

【より素早くスムーズな追跡のためのヒント】

ロボットの真上から見たときのカメラ、ボール、スクリーンの関係を図 32 に示す。図中の θ がカメラから見たボールのある水平方向角度となる。θ をどのように計算するか、その値をどう利用するか考えてみよ。

ソースコード 4: ballsearch_active_run()

```
void SouseiDefault::ballsearch_active_run() {
  cout << "-------------------------------------" << endl;
  cout << "Detect red ball and move head." << endl;
  cout << "DO NOT EXECUTE THIS ON REAL ROBOT !!" << endl;
  cout << "-------------------------------------" << endl;

  //Camera On
  enableCamera();

  // step
  myStep();

  // current position
  double pan = getPosition(Neck);
  double tilt = getPosition(Head);

  while (true) {
    double red_x, red_y;
    bool rFound = getRedBallCenter(red_x,red_y);
    if( rFound == true ) {
        cerr << "red ball found! << (" << red_x << "," << red_y <<")" << endl;
    }
```

```
    if( rFound )
        setEyeLEDColor(0xff0000);
    else
        setEyeLEDColor(0x000000);

    double range = 0.5; // too sensitive
    double step_deg = 8.0; // too large update

    double step = deg2rad(step_deg);
    if( rFound ) {
      // Pan
      if(red_x > (1.-range) * getImageWidth())
        pan -= step;
      else if(red_x < range * getImageWidth())
        pan += step;
      // Vertical
      if(red_y > (1.-range) * getImageHeight())
        tilt -= step;
      else if(red_y < range * getImageHeight())
        tilt += step;
    }
    setPosition(Neck,pan);
    setPosition(Head,tilt);

    // step
    myStep();
  }
}
```

11 課題５：ボールを落としあう対戦用プログラムの実装

課題４で赤いボールを画像センサを用いて検知し、ボールが画面の中央に位置するように首関節 (Neck,Head) を制御するプログラムを作った。そのコードを用いればボールがロボットの体に対してどちらの方向にあるのかがわかることになる。課題５では、この情報を用いることで向かい合って立つ相手方ロボット (敵機) の保持しているボールに向けて自分のロボット (自機) の手を移動させ、ボールを落とす、というゲームのプレイヤーとしての行動を実現する (図 33)。ゲームに勝利するには敵機のボールを素早く落とす (図 34) だけでなく、敵機の攻撃から自機のボールを守る動作 (図 35) も重要となる。

課題５は 5 - 1 から 5 - 7 の部分課題に分かれており、各部分課題でそれぞれロボットに機能を追加していくことで対戦ロボット用プログラムを作成する (図 36)。各部分課題で追加する機能を以下に示す。詳細については対応する節を参照すること。

演習課題５- １ ボールの方向の把握 (11.2 を参照)

`fighter-single.wbt` で動作を確認する。

演習課題５- ２ ボールの見えている大きさの把握 (11.3 を参照)

`fighter-single.wbt` で動作を確認する。

演習課題５-３ 状態遷移による攻撃と戻し動作の切り替え (11.4 を参照)
`fighter-single.wbt` で動作を確認する。

演習課題５-４ 接触による動作の変更 (11.5 を参照)
`rival-doll.wbt` で動作を確認する。

演習課題５-５ 接触相手に衝撃を与えない動作 (11.6 を参照)
`rival-doll.wbt` で動作を確認する。

演習課題５-６ 発見したボール位置への手先の移動 (11.7 を参照)
`fighter-single.wbt` で動作を確認する。

演習課題５-７ 対戦で各レベルの敵機に勝てる動作 (11.8 を参照)
対戦用の環境 `rival-level*.wbt` で動作を確認する。

各部分課題で用いる環境を図 37 に示す。
最終的に、あらかじめ用意されたプログラムを組み込んだ敵機と対戦を行い、それに勝てる動作の実装を目指す。対戦の基本的なルールは以下の通りである。

- 自機と敵機は真正面で向かい合う形で設置する。両者は膝と膝の間の距離が 23[cm] となるように設置する。
- 自機と敵機はそれぞれ、左手にボールの台、右手に武器を装着し台にボールを載せた状態で対戦を開始する。
- 対戦時の動作開始は赤もしくは青のフラッグ (旗) で指示する。ロボットの視界をフラッグで覆った後、フラッグを取り去った時点で動作を開始する。
- 自機のボールを落とさずに敵機のボールを落とせば自機の勝利とする。但し敵機自体を揺らすなど敵機のボールに触れずに落とした場合は勝利と見なさず、引き分けとする。
- 敵機の武器や台、ロボット本体などに触れて大きく動かした場合はその時点で敗北とする。

課題５で用いる wbt ファイルでは、向かって左側に自機のロボットが設置され、向かって右側にボール、もしくは敵機のロボットが設置されている。ロボットにはそれぞれ右手に攻撃用に延長した手先 (武器)、左手にボールを載せる台が装着されている。シミュレーションを開始するとロボットは待機状態に入り、フラッグが移動するのを待って動作を開始する。
課題５-７では自機のボールが落とされないようにしつつ敵機が左手の台に載せているボールを落とせるよう自機のプログラムを実装する。自機と敵機で数回の対戦を行い、その結果で実装を評価する。

11.1 サンプルプログラムの実行

課題５ではロボットの右手に武器、左手にボール台が装着された環境を用いる。課題５で用いる各 wbt ファイルの内容は以下の通りである。

`fighter-single.wbt`
武器とボール台を装着した自機ロボットと、空中に浮遊するボールのみがある環境。ボールを特定の位置に置いたときのカメラ上の見え方を確認できる。

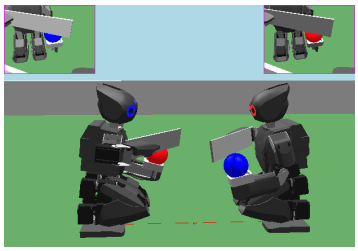


図 33: 対戦ゲームの状況 (向かって左が自機、右が敵機)

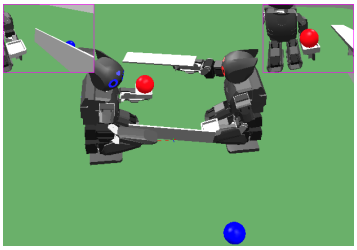


図 34: 先に相手のボールを落とせば勝利

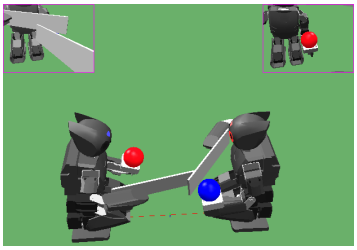


図 35: 相手の攻撃を防御することも重要

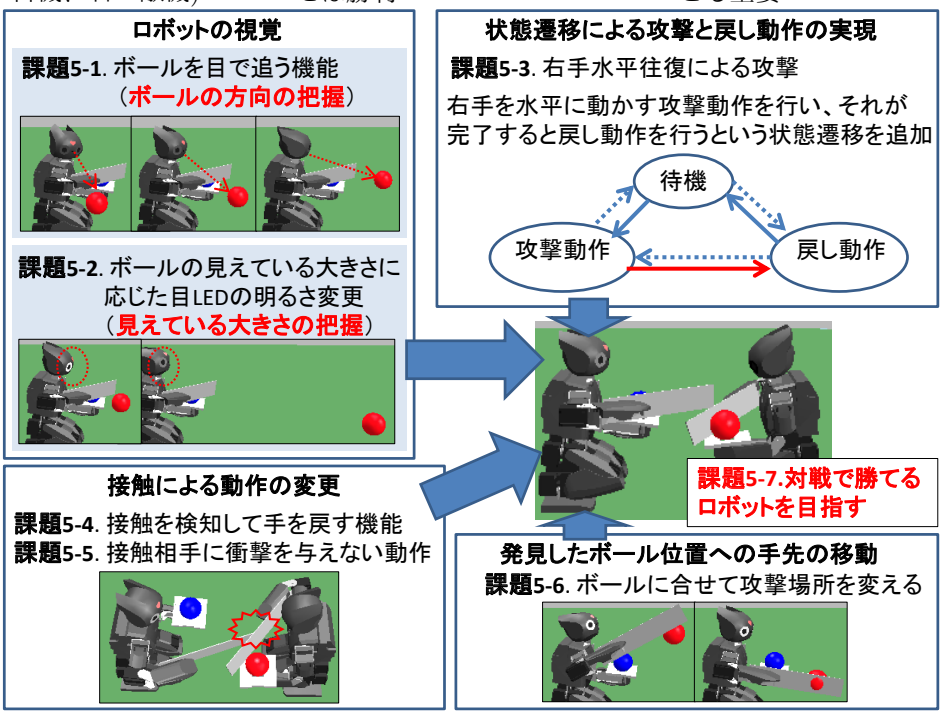


図 36: 課題 5 の部分課題の概要

`fighter-check-pose.wbt`

関節角度とロボットの姿勢の関係を確認するための環境。使い方は 11.7 の囲みを参照すること。

`rival-doll.wbt`

敵機の姿勢を自由に変更できる環境。敵機が特定の姿勢にあるときの自機の動作を確認できる。使い方は 11.7 の囲みを参照すること。

`rival-level0.wbt`, `rival-level1.wbt`,... `rival-level5.wbt`

敵機がそれぞれレベル 0 からレベル 5 の動作を行う環境。この環境で自機のボールを守りつつ敵機のボールを落とせるよう自機の行動を実装する。

課題 5 では関数 `attack_and_defend()` (ソースコード 7) に機能を追加して対戦に勝てるロボットプログラムの構築を目指す。

この関数 `attack_and_defend()` は画像からボールの位置を取得して首の向きを制御しつつ、状況に応じて手を移動させるプログラムのひな形になっている。サンプルプログラムの流れを図 38 に示す。

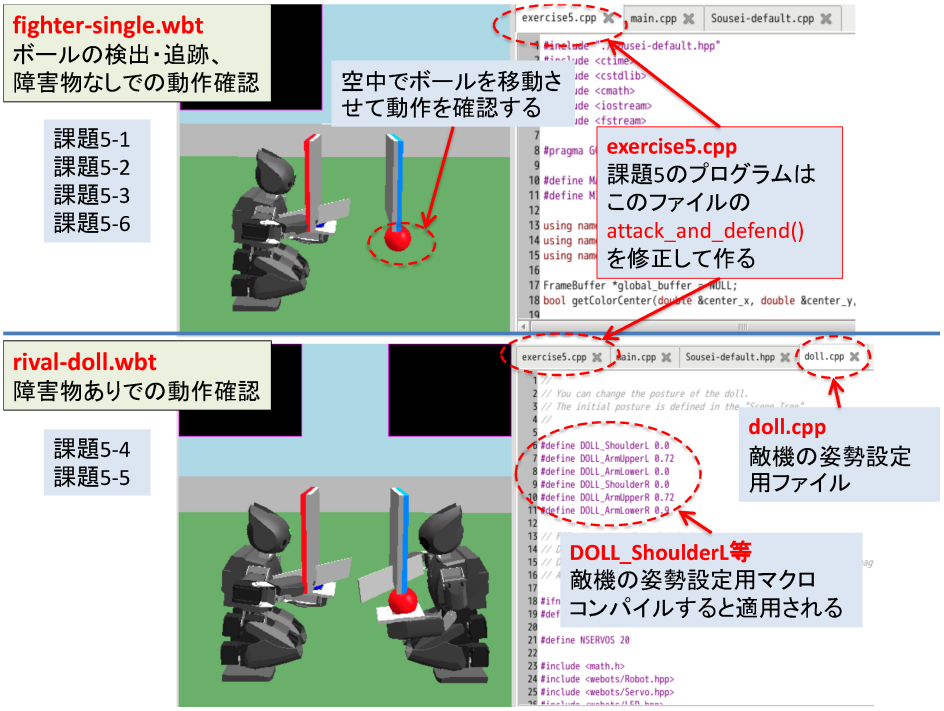


図 37: 課題 5 の部分課題で用いる環境

`getPosition()` での現在のモーター角度の取得や `setPosition()` での目標角度の設定の処理はあらかじめ組み込んであるので、課題 5 においては現在のモーター角度やトルク (`current_lpos0`,`force_lpos0` など) から次に目指すべき目標角度 (`lpos0` など) を計算する部分 (8.2 から 8.5) を中心に追加・修正することとなる。

1. から 8. の各項目で行われる処理は以下の通りである。各項目はプログラム中のコメントと対応している。

1. INITIALIZE THE ROBOT.
ロボットを初期化する。

2. DEFINE CONSTANTS.
ロボットに関する定数を定義する。

3. DECLARE MODES AND THRESHOLDS. 動作を表す列挙型と動作切り替えのための閾値を設定する (動作切り替えについては 11.4 を参照)。

4. PREPARE VARIABLES. 変数を宣言し、初期化する。

5. INITIALIZE THE POSE OF THE ROBOT.
関節角度を設定してロボットに初期姿勢を取らせる。

6. WAIT UNTIL THE CURTAIN IS FOUND.
試合開始を告げるカーテンが見つかるまで待機する。カーテンが見つかるとカーテンと同じ色のボールを攻撃するよう色の設定を行う。

7. WAIT UNTIL THE CURTAIN IS REMOVED.
カーテンが視界から消えて、試合開始となるまで待機する。

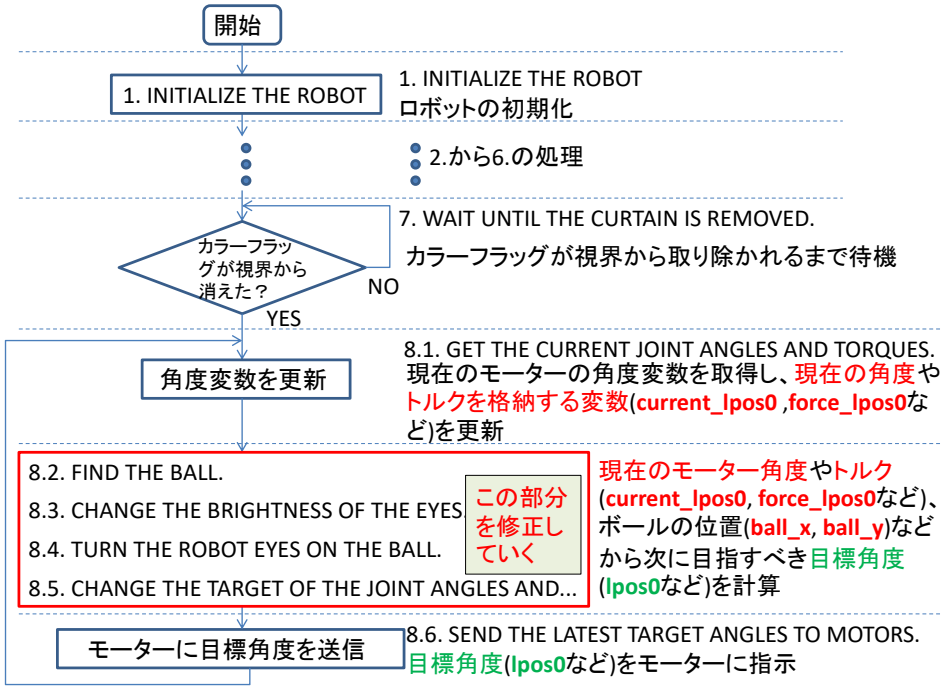


図 38: `attack_and_defend()` の流れ

8. START THE GAME.

無限ループで試合開始後の動作を行う。ループ内部の構成は以下の通り。

8.1 GET THE CURRENT JOINT ANGLES AND TORQUES.

各モーターについて現在の関節角度 (`current_lpos0` など) とトルク (`force_lpos0` など) を取得して変数に代入する。

8.2 FIND THE BALL.

カメラでボールを検出し、その位置と面積を変数に代入する (ボールの情報取得については 11.3 を参照)。

8.3 CHANGE THE BRIGHTNESS OF THE EYES.

カメラ画像上のボールの面積に応じて目の LED の明るさを変更する (LED の明るさ変更については 11.3 を参照)。

8.4 TURN THE ROBOT EYES ON THE BALL.

視野の中心にボールが来るよう、自機の首のモーターの目標関節角度を変更する (課題 4 の内容)。

8.5 CHANGE THE TARGET OF THE JOINT ANGLES AND UPDATE...

現在行っている動作 (待機、攻撃動作など) に応じた目標関節角度の設定を行う。ボールを発見すると待機から攻撃動作に移るなど、必要であれば動作を切り替える。(動作の切り替えについては 11.4 を参照)

8.6 SEND THE LATEST TARGET ANGLES TO MOTORS.

現在の目標関節角度 (`lpos0`) をモーターに通知し、時間を進める。

11.2 ボールの追跡

【演習課題５-１】

サンプルプログラムの attack_and_defend() 関数を改造し、課題４のプログラムを組み込んで敵機のボールを目で追う処理を追加せよ。

11.3 ボールの見えている部分の面積の取得

課題５においては敵機のボールを隠す行動のため、ボールの一部が見えない場合がある。敵機がボールの前に腕を移動させて隠している場合はボールの一部が見えていても、攻撃時には敵機の腕に接触してボールに触れられない恐れがある。したがって、ボールが見えているかだけでなく、どの程度見えているかに応じて自機の動作を変更する必要がある。

ボールのうちの見えている部分の面積 (画素数) は getColorCenter() 関数を呼び出せば取得できる。課題５のサンプルプログラムではソースコード５のように getColorCenter() 関数を呼び出して、ボールの見えている部分の面積 (画素数) を ball_area という変数に代入している。

ソースコード 5: ボールの位置と面積の取得

```
ball_is_found = getColorCenter(ball_x, ball_y, ball_area, mCamera,
    color_info);
```

この面積を用いれば、ボールの大部分が見えているときだけ攻撃するといった動作が可能となる。シミュレータと実機ではカメラの解像度が異なるため、ボールの面積だけでなくボールが視野全体に占める割合を考える必要があることに注意する。

【演習課題５-２】

attack_and_defend() 関数を改造し、視野内でボールが大きく映っているほど目の LED を明るくする処理を追加せよ。

シミュレータ上でボールと自機との距離に応じて LED の明るさが変化することを確認せよ。シミュレータで正常な動作を確認したら、実機で動作させてその様子を観察せよ。

サンプルプログラムではボールを視野内に発見した場合に目の LED を brightness 変数に対応した明るさにする処理が実装されているのでボールの面積 (ball_area 変数) から brightness 変数の値を計算する処理を追加すればよい。brightness 変数は 0 から 255 の範囲でなければならないため、まず面積の値を cout 等で出力して面積としてあり得る範囲を確認して brightness の計算処理を実装すること。

11.4 状態遷移による攻撃と戻し動作の実現

敵機のボールを落とすためにはカメラで発見した敵機のボールの方向へ自機の手を動かす必要があるが、その途中で敵機に接触した場合、敵機を押してしまうことのないよう、ある程度腕を戻さなければならない。この戻す動作の最中に敵機のボールの方向に腕を動かすと再度接触する可能性が高いため、戻す動作の際には敵機のボールを無視しなければならない。このように課題５においては過去の状況も考慮して次の動作を決定する必要がある。

サンプルプログラムでは実行中の動作として待機、攻撃動作、手を戻す動作の 3 種類を考え、そのそれぞれを表す列挙型 ACTION_MODE を導入している。列挙型 ACTION_MODE には 3 種類の動作に対応して

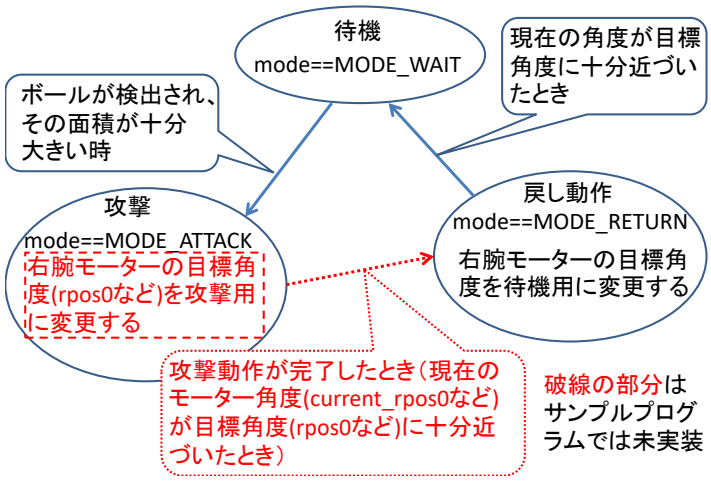


図 39: 状態遷移

MODE_WAIT(待機中)、MODE_ATTACK(攻撃動作中)、MODE_RETURN(戻し動作中) という 3 つの識別子を定義しており、変数 mode に現在の動作を表す識別子を代入するようになっている。サンプルプログラムに定義された状態の遷移関係を図 39 に示す。図中の破線で示された部分はサンプルプログラムには実装されていないので課題 5 - 3 ではこの部分を追加実装する。

サンプルプログラムの動作切り替えの流れは以下のようになっている。

1. プログラム開始時は mode を MODE_WAIT とする。
2. while ループ内で以下の処理を繰り返す。

   mode が MODE_WAIT の時 敵機ボールを発見し、それが視野内で十分大きく (シミュレータ上で 600[pixel] 以上) 映っていれば攻撃動作に入ることとし mode を MODE_ATTACK に変更する。

   mode が MODE_ATTACK の時 攻撃動作のために目標関節角度を変更する (サンプルプログラムでは未実装)。現在の関節角度と目標関節角度を比較し両者が十分近い場合は戻し動作に入ることとし、mode を MODE_RETURN に変更する (サンプルプログラムでは未実装)。

   mode が MODE_RETURN の時 腕が初期姿勢を取るよう目標関節角度を変更する。現在の関節角度と目標関節角度を比較し両者が十分近い場合は待機動作に入ることとし mode を MODE_WAIT に変更する。

この流れを図 40 に示す。サンプルプログラムは待機中にボールを発見すると攻撃動作に移るが攻撃動作、戻し動作の最中はボールを無視して動作を続行する。

サンプルプログラム上では動作の種類を 3 つとして列挙型を宣言しているが、これはあくまで例であるので 3 つで十分とは限らない。敵機を押したり、自機のボールを落とされたりすることなく敵機のボールを落とすためにはどのような動作が必要か、またそれらはどういう条件で切り替えるべきかを考え、必要に応じて追加すること。

【演習課題 5 - 3】

サンプルプログラムでは攻撃動作時の目標関節角度の設定する処理と、攻撃動作が完了した際に戻し動作に移る処理が実装されていない (図 39、40 の破線部分)。attack_and_defend() 関数を改造し、攻

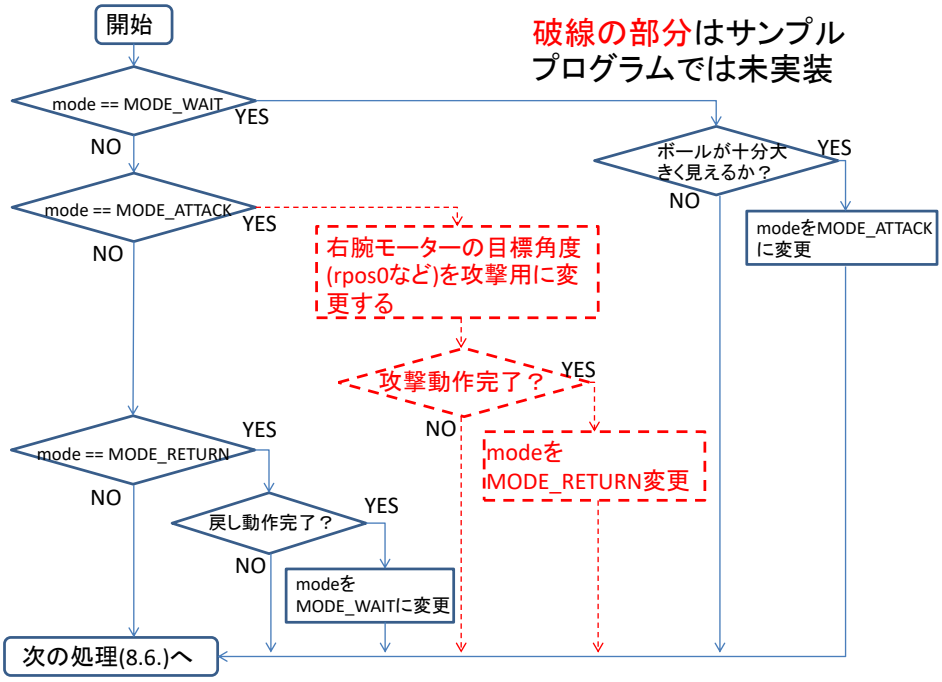


図 40: 状態遷移を実現するプログラムの流れ

撃動作として右手先を水平方向に右に動かす処理と、右手先が自機の正面を向いた時点で攻撃動作を完了し戻し動作に移る処理を追加せよ。

シミュレータ上で動作が確認できたら実機でも動作させ、ボールを見つけるたびに右手を水平に往復させることを確認せよ。その様子を記録しレポートに報告すること。

11.5 手先の接触検知

この課題においては、敵機の腕など、ボール以外の部分に触れて敵機本体を動かすと敗北となるため、武器がボール以外のものに接触した場合は手先を手元に戻すなどの対処が必要となる。

自機の右腕が敵機の腕や本体など、重い物体に接触して抵抗を受けると、その抵抗に応じた大きさのトルクが右腕にかかる。そこで、右腕のモータ (特に肘関節) にかかっているトルクをチェックし、強いトルクがかかった場合には攻撃動作の状態から戻し動作の状態に移るよう状態遷移を追加すれば良い (図 41 の破線部分)。

各関節にかかっているトルクは `getMotorForceFeedback()` 関数によって取得できる。この関数は引数で指定したモータにかかっているトルクを $[\mathrm{N \cdot m}]$(Newton meter) 単位で表した値を返す。

サンプルプログラムでは `getMotorForceFeedback()` 関数を用いて各モーターにかかっているトルクを求め、その値を `force_rpos0` などの変数に代入している (図 38 の 8.1、ソースコード 7 の 8.1 の部分)。例えば右肘のモーター (`ArmLowerR`) にかかっているトルクの値は `force_rpos2` 変数に代入される。

`rival-doll.wbt` の環境では `doll.cpp` が開かれて `doll.cpp` の冒頭にある `DOLL_ShoulderL` などのマクロ群の定義を変更してコンパイルすれば、敵機 (向かって左側のロボット) の姿勢を自由に変更できる。

自機の右手往復運動の途中で敵機の腕が接触するよう敵機の姿勢を調整し、障害物接触時に各モーターのトルクの値がどのように変化するか確認せよ。接触時のモーターのトルクの値をもとにして、接触し

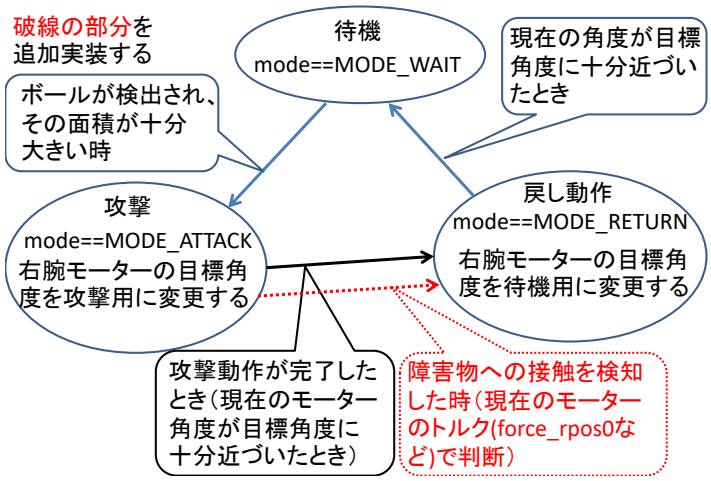


図 41: 接触検知による状態遷移

たと判断する条件を定めればよい。

【演習課題５-４】

`attack_and_defend()` 関数を改造し、右手先を水平方向に往復するよう動かし、攻撃動作中に手先に障害物が接触すると戻し動作に移って回避するという処理を追加せよ。

シミュレータの `rival-doll.wbt` の環境で敵機に接触させ動作が確認せよ。動作が確認できたら実機でも動作させ、手で保持したボールを障害物として接触させ、その動作を観察せよ。

11.6　接触相手に衝撃を与えない動作

これまでの課題では `setPosition()` 関数によって関節角度を変更していたが、この関数はデフォルトではモーターの出せる最高速度で関節角度を変更する。このため、単純にこれを用いると手先が非常に速く動いて、接触を検知した際には既に敵機に大きな衝撃を与えて動かしてしまっていて、敗北となることが起こりやすい。そこで敵機に大きな衝撃を与えないように、また敵機を動かしてしまう前に動作を修正できるように、腕の動作速度を調整する必要がある。

`setPosition()` 関数で関節角度を変更する際の角速度は `setVelocity()` 関数で変更できる。この関数の第 1 引数はモーターの ID で、第 2 引数は角速度を [rad/sec](radian 毎秒) で表した値である。例えばソースコード 6 のように `setVelocity()` 関数を呼び出すと右肘 (`ArmLowerR`) の角速度が 30[deg/sec] = 0.524[rad/sec] となる。

ソースコード 6: モーター速度設定関数

```
setVelocity(ArmLowerR, deg2rad(30))
```

`setVelocity()` 関数で設定を行わない場合の角速度 (モーターの最高速度) は約 700[deg/sec] である。

【演習課題５-５】

【演習課題５-４】で作成した `attack_and_defend()` 関数にモーターの速度制限を導入し、手先に障害物が接触したときに大きな衝撃を与えず戻し動作に移れるよう改造せよ。

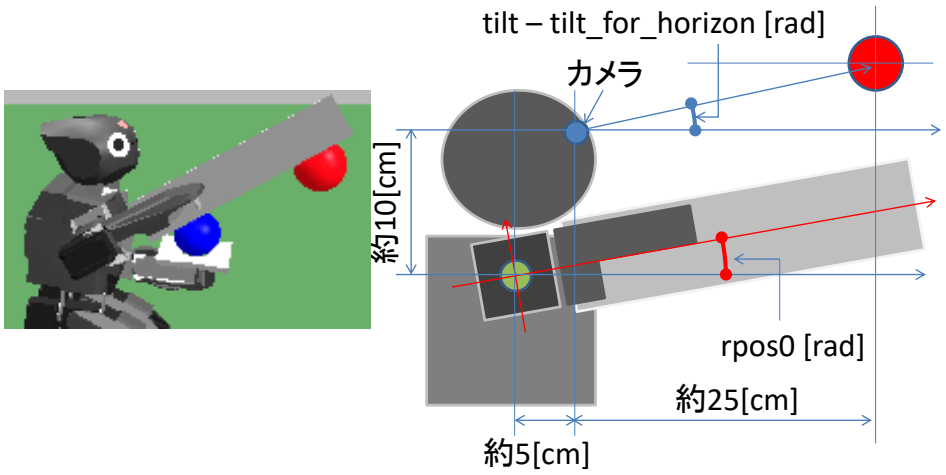


図 42: ボールに触れるための上下方向の角度

シミュレータの rival-doll.wbt の環境で敵機に接触させ (敵機を動かすほどでない) 弱いトルクの接触で動作が反転することを確認せよ。動作が確認できたら実機でも動作させ、手で保持したボールを障害物として接触させ、その動作を観察せよ。

11.7 首関節の値を用いたボールの存在する位置・方向の推定と手の誘導

敵機のボールが見えているときは常にボールが画像の中央に位置するように制御出来ているとすると、Neck 関節の値（pan 変数）、Head 関節の値（tilt 変数）によってボールの存在する方向を推定できる。また、自機と敵機の位置は固定されているため、方向が分かればおおまかな 3 次元位置も推定できる。

敵機のボールを落とすため推定した位置に手先を移動させるにはどのように関節を制御すればよいかが課題 5 の重要なポイントである。

首が上を向いている（Head 関節の値（tilt 変数）が大きい）場合には敵機のボールが自機のカメラよりも高い位置にあるので手先を上に移動させ、逆に首が下を向いている（Head 関節の値（tilt 変数）が小さい）場合は逆に手先を下に移動させればよい。

一例として右脇 (ArmUpperR 関節) を直角に開いた場合のロボットとボールの位置関係を図 42 に示す。図中の tilt_for_horizon はカメラが水平方向を向いているときの tilt の値であり、サンプルプログラム中で定義済みの定数である。手先をボールに触れさせるには右肩 (ShoulderR) の角度（rpos0 変数）をどのように設定すればよいか考えよ。

水平方向についても同様で、首が大きく右を向いている（Neck 関節の値（pan 変数）が負の値で絶対値が大きい）場合は手先を大きく右に移動させ、首が正面に近い方向を向いている（Neck 関節の値（pan 変数）の絶対値が小さい）場合は手先を正面に近い方向に移動させる。このようにすれば敵機のボールの位置に応じて手先が移動させることができる。

【演習課題 5 - 6 】

attack_and_defend() 関数を改造し、止まっているボールを見つけてその位置に手先を移動しボールを落とす処理を追加せよ。攻撃時の手先の姿勢や軌道は課題 5 - 3 の水平移動に限らず自由に設定してよい。敵機の姿勢を自由に変更できる環境 rival-doll.wbt (下記のヒントを参照) を用い、ボールの位置

を様々に変更して試行すること。シミュレータでボールの様々な位置に対応して動作することが確認できればプログラムを実機に転送し、実機の前にボールを提示してそれに触れることができるか確認せよ。

カメラ画像の情報からボールの方向と 3 次元位置を推定する方法や推定した位置に手先を移動するための方法と動作結果について、レポートに報告すること。

【色々な姿勢での攻撃を実現するためのヒント】

> 手先の位置は腕に 3 つある関節全てに依存しているため、手先を前後左右指定した方向に動かすためには 3 つの関節角度をまとめて変更する必要がある。

ある手先の 3 次元位置に対してそれを実現する 3 つの関節角度の組み合わせを計算する (逆問題) のは難しい。そこで 3 つの関節角度を様々に変更して所望の位置に手先を移動する角度をあらかじめ見つけておき、その関節角度を再生することによって手先を所望の位置に移動させる方法が考えられる。単純な方法としては関節角度の組み合わせをあらかじめ複数求めておき、それらの中から手先がボールに最も近くなるものを用いる方法がある。しかし、あらゆる位置についての関節角度を手動で設定するのは手間がかかるので、少数の位置についてだけ手動で設定しておき、その他の位置については自動的な計算で補う工夫が考えられる。

シミュレータ内での動作確認用 wbt ファイル

課題 5 用に追加配布しているサンプルプログラムキット (`darwinop-sousei-d1d3-ex5.zip`) には**腕の関節角度と手先位置の関係**を確認するための wbt ファイル `fighter-check-pose.wbt` や**敵機の姿勢を自由に変更できる** wbt ファイル `rival-doll.wbt` やが含まれている。これらのファイルに収められた環境を用いて関節角度と手先位置の関係やカメラ画像とボール位置の関係を把握しておけば、カメラ画像からボール位置を推定したり、特定の位置にあるボールに触れるよう手先を移動させたりする際に活用できる。これらの環境を用いる際には、wbt ファイルを開いた後に一度コンパイルボタンを押して実行環境を整える必要がある。
それぞれの環境の使い方は以下の通りである。

`fighter-check-pose.wbt`

wbt ファイル `fighter-check-pose.wbt` 内の環境を用いることで**腕の関節角度と手先位置の関係**を確認できる。この環境ではシミュレーション実行時にロボットの前面に置かれた 6 色のスライダーをロボットの左右方向に動かすことでロボットの腕関節角度を変更できる。変更後の右腕の関節角度は Webots のコンソールに出力されるので、斜面に対して適切な位置に腕を移動させるための関節角度を知ることができる (図 43)。

`rival-doll.wbt`

wbt ファイル `rival-doll.wbt` 内の環境を用いることで**敵機が特定の姿勢にあるときの動作**を確認できる。この環境では自機の行動を表すプログラムファイル `Sousei-default.cpp` に加えて `doll.cpp` が開かれている。`doll.cpp` の冒頭にある `DOLL_ShoulderL` などのマクロ群の定義を変更してコンパイルすれば、敵機 (向かって右側のロボット) の姿勢を自由に変更できる。この環境を用いることで、敵機が特定の姿勢にある場合にロボットのカメラにどのように映るのか、どういう動作を行うかを確認できる。

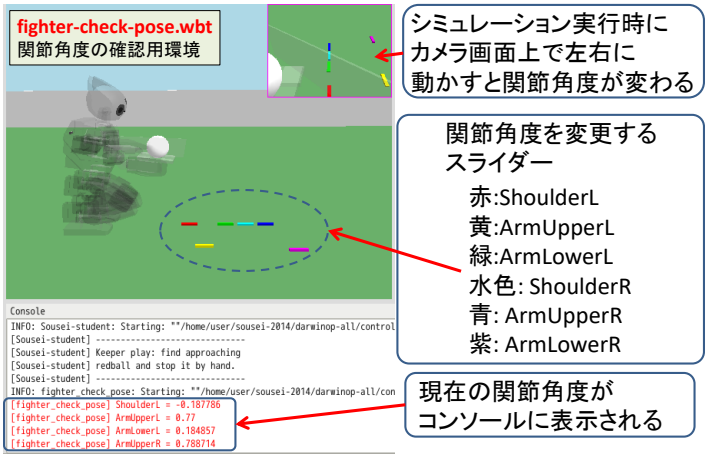


図 43: `fighter-check-pose.wbt`

11.8 対戦に勝つための動作

【演習課題５-７】

配布されたファイルに含まれる各レベルの敵機と対戦を行い、それに勝てるプログラムを作成せよ。攻撃や防御の姿勢は自由に変更してよい。

各レベルの敵機と対戦するための wbt ファイルと敵機の行動は以下の通りである。

レベル 1 (`rival-level1.wbt`) ボールを上下方向ランダムな位置に移動させた後、静止する。ボールを隠すなどの防御は行わない。

レベル 2 (`rival-level2.wbt`) ボールを上下に移動させ続ける。ボールを隠すなどの防御は行わない。

レベル 3 (`rival-level3.wbt`) ボールの前で右手先を上下させ、ボールを隠したり見せたりする。ボール自体は動かさない。

レベル 4 (`rival-level4.wbt`) ボールを上下に移動させつつ、その前で右手先を上下させ、ボールを隠したり見せたりする。

レベル 5 (`rival-level5.wbt`) ボールを右手先で隠し、相手のボールを発見すると攻撃を行う。

サンプルコード `attack_and_defend()` を以下の順に改造するとよい。

1. 推定した位置に手先を移動させボールを落とす攻撃動作を組み込む。(レベル 1 の敵機に勝利できるようにする)
2. 移動するボールに合わせて移動先を変えつつ攻撃する処理を組み込む。(レベル 2 の敵機に勝利できるようにする)
3. ボールを隠しての防御と、防御を解除してボールを見せる動作を繰り返す敵機に対して、防御が解けたことを検知しそのタイミングで攻撃する処理を組み込む。(レベル 3 の敵機に勝利できるようにする)
4. ☆敵機のボールの一部が隠されている状況でも敵機のボールを落とせるよう工夫した攻撃処理を組み込む。(レベル 4 の敵機に勝利できるようにする)

5. ☆自機のボールへの攻撃を防御しながら敵機のボールを落とす処理を組み込む。(レベル 5 の敵機に勝利できるようにする)

このとき、実際にシミュレータや実機で動作させてみるとさまざまな工夫が必要となることがわかる。レポートにはどのようなアルゴリズムではどのような問題が発生し、それをどう分析し、どういう工夫によって解決したか、順にすべて報告せよ。プログラムコードは最終版を添付せよ。シミュレータで十分試してから、実機でも動作させて有効性を評価せよ。

ソースコード 7: attack_and_defend()

```
void SouseiDefault::attack_and_defend() {
  // ============================================================
  // 1. INITIALIZE THE ROBOT.
  //

  // LED
  setHeadLEDColor(0x00FF00);

  myStep();
  enableCamera();
  enableArmMotorForceFeedback();
  cout << "Image: (" << getImageWidth() << "x"
       << getImageHeight() << ")" << endl;
  myStep();

  // additional initialization
  mFoundBlueCurtain = new DARwInOPVisionManager
    (getImageWidth(), getImageHeight(), 210, 30, 20, 10, 50, 100);


  // ============================================================
  // 2. DEFINE CONSTANTS.
  //
  double view_w_angle = deg2rad(58.0);
  //double view_h_angle = deg2rad(view_w_angle*3./4.);
  double focal_length = getImageWidth()/2. / tan(view_w_angle/2.);

  // parameter from 'rotation' of 'DHead' in 'protos/DARwInOP-with-ball.proto
      '.
  const double tilt_for_horizon = 0.7854;



  // ============================================================
  // 3. DECLARE MODES AND THRESHOLDS.
  //
  enum ACTION_MODE { MODE_WAIT = 0, MODE_ATTACK = 1, MODE_RETURN = 2 };
  int mode = MODE_WAIT;

  //
  // The threshold of the area.
```

```
// Note that the resolution of the camera on the simulation
// may differ from that on the real robot.
//
double ball_area_threshold = 600.0 * (getImageWidth() / 160.0) * (
    getImageHeight() / 120.0);



// ============================================================
// 4. PREPARE VARIABLES.
//
const char *role_name_array[2] = {"RED EYES", "BLUE EYES"};
const char *role_name = NULL;
int seed = time(NULL);
ColorFinder color_info;
bool with_red_eyes = true;
bool curtain_is_found = false;
bool red_curtain_is_found = false;
bool blue_curtain_is_found = false;

// The initial angles.
double lpos0_initial = 0, lpos1_initial = 0, lpos2_initial = 0;
double rpos0_initial = 0, rpos1_initial = 0, rpos2_initial = 0;

// The current target angles.
double lpos0 = 0, lpos1 = 0, lpos2 = 0;
double rpos0 = 0, rpos1 = 0, rpos2 = 0;
double pan = 0, tilt = 0;

// Variables for tracking the ball.
double step_pan = 0, step_tilt = 0;
int missing_count = 10;
int visible_count = 0;

// The ranges of angles.
double lpos0_min = getMinPosition(ShoulderL);
double lpos1_min = getMinPosition(ArmUpperL);
double lpos2_min = getMinPosition(ArmLowerL);
double lpos0_max = getMaxPosition(ShoulderL);
double lpos1_max = getMaxPosition(ArmUpperL);
double lpos2_max = getMaxPosition(ArmLowerL);
double rpos0_min = getMinPosition(ShoulderR);
double rpos1_min = getMinPosition(ArmUpperR);
double rpos2_min = getMinPosition(ArmLowerR);
double rpos0_max = getMaxPosition(ShoulderR);
double rpos1_max = getMaxPosition(ArmUpperR);
double rpos2_max = getMaxPosition(ArmLowerR);

double pan_min = getMinPosition(Neck);
double pan_max = getMaxPosition(Neck);
```

```
  double tilt_min = getMinPosition(Head);
  double tilt_max = getMaxPosition(Head);


  // ============================================================
  // 5. INITIALIZE THE POSE OF THE ROBOT.
  //
#ifdef CROSSCOMPILATION
  // for a real robot.
  cout << "CROSS COMPILATION" << endl;
  lpos0_initial = 0.0;
  lpos1_initial = 0.72;
  lpos2_initial = 0.0;
  rpos0_initial = 0.0;
  rpos1_initial = 0.72;
  rpos2_initial = 0.90;
#else
  // for simulation
  cout << "SIMULATION" << endl;
  lpos0_initial = getPosition(ShoulderL);
  lpos1_initial = getPosition(ArmUpperL);
  lpos2_initial = getPosition(ArmLowerL);
  rpos0_initial = getPosition(ShoulderR);
  rpos1_initial = getPosition(ArmUpperR);
  rpos2_initial = getPosition(ArmLowerR);
#endif

  setPosition(ShoulderL, lpos0_initial);
  setPosition(ArmUpperL, lpos1_initial);
  setPosition(ArmLowerL, lpos2_initial);
  setPosition(ShoulderR, rpos0_initial);
  setPosition(ArmUpperR, rpos1_initial);
  setPosition(ArmLowerR, rpos2_initial);
  lpos0 = lpos0_initial;
  lpos1 = lpos1_initial;
  lpos2 = lpos2_initial;
  rpos0 = rpos0_initial;
  rpos1 = rpos1_initial;
  rpos2 = rpos2_initial;


  // ============================================================
  // 6. WAIT UNTIL THE CURTAIN IS FOUND.
  //
  while (true) {
    double curtain_x = 0, curtain_y = 0;
    if(curtain_is_found) {
      break;
    }
    else {
```

```
    red_curtain_is_found = getRedCurtainCenter(curtain_x, curtain_y);
    blue_curtain_is_found = getBlueCurtainCenter(curtain_x, curtain_y);
    curtain_is_found = red_curtain_is_found || blue_curtain_is_found;
  }
  // step
  myStep();
}

// Change the color of eyes according to the color of the curtain.
if(red_curtain_is_found) {
  with_red_eyes = true;
  setHeadLEDColor(0xff0000);

  // int m_hue; /* 0 ~ 360 */
  // int m_hue_tolerance; /* 0 ~ 180 */
  // int m_min_saturation; /* 0 ~ 100 */
  // int m_min_value; /* 0 ~ 100 */
  // double m_min_percent; /* 0.0 ~ 100.0 */
  // double m_max_percent; /* 0.0 ~ 100.0 */
  color_info.m_hue = 0;
  color_info.m_hue_tolerance = 15;
  color_info.m_min_saturation = 30;
  color_info.m_min_value = 15;
  color_info.m_min_percent = 0;
  color_info.m_max_percent = 30;
}
else {
  with_red_eyes = false;
  setHeadLEDColor(0x0000ff);

  // int m_hue; /* 0 ~ 360 */
  // int m_hue_tolerance; /* 0 ~ 180 */
  // int m_min_saturation; /* 0 ~ 100 */
  // int m_min_value; /* 0 ~ 100 */
  // double m_min_percent; /* 0.0 ~ 100.0 */
  // double m_max_percent; /* 0.0 ~ 100.0 */
  color_info.m_hue = 240;
  color_info.m_hue_tolerance = 15;
  color_info.m_min_saturation = 40;
  color_info.m_min_value = 10;
  color_info.m_min_percent = 0;
  color_info.m_max_percent = 30;
}
if(with_red_eyes) {
  role_name = role_name_array[0];
}
else {
  role_name = role_name_array[1];
}
cerr << role_name << ": SEED = " << seed << endl;
```

```
// ============================================================
// 7. WAIT UNTIL THE CURTAIN IS REMOVED.
//
while (true) {
  bool found_now = false;
  double curtain_x = 0, curtain_y = 0;
  found_now
    = (red_curtain_is_found && getRedCurtainCenter(curtain_x, curtain_y))
    || (blue_curtain_is_found && getBlueCurtainCenter(curtain_x, curtain_y));

  if(!found_now) {
    break;
  }

  // step
  myStep();
}



// ============================================================
//
// 8. START THE GAME.
//
//
while (true) {
  // ============================================================
  // 8.1 GET THE CURRENT JOINT ANGLES AND TORQUES.
  //
  double current_pan = getPosition(Neck);
  double current_tilt = getPosition(Head);
  double current_lpos0 = getPosition(ShoulderL);
  double current_lpos1 = getPosition(ArmUpperL);
  double current_lpos2 = getPosition(ArmLowerL);
  double current_rpos0 = getPosition(ShoulderR);
  double current_rpos1 = getPosition(ArmUpperR);
  double current_rpos2 = getPosition(ArmLowerR);

  // Collect torques of motors.
  // The unit is [N * m] (Newton meter).
  // http://www.cyberbotics.com/dvd/common/doc/webots/reference/section3.47.
      html#wb_motor_enable_force_feedback
  double force_lpos0 = getMotorForceFeedback(ShoulderL);
  double force_lpos1 = getMotorForceFeedback(ArmUpperL);
  double force_lpos2 = getMotorForceFeedback(ArmLowerL);
  double force_rpos0 = getMotorForceFeedback(ShoulderR);
  double force_rpos1 = getMotorForceFeedback(ArmUpperR);
```

```
double force_rpos2 = getMotorForceFeedback(ArmLowerR);


// ============================================================
// 8.2 FIND THE BALL.
//

// getColorCenter() returns the following information.
//
// (ball_x, ball_y) : the position of the found ball.
//
// ball_area : the area of the visible region of the ball.
// [pixel]
//
// ball_is_found : true if the ball is found.
// false if the ball is not found.
//
double ball_x = 0, ball_y = 0, ball_area = 0;
bool ball_is_found = false;
ball_is_found = getColorCenter(ball_x, ball_y, ball_area, mCamera,
    color_info);

if(ball_is_found == true) {
  missing_count = 0;
  visible_count++;
}
else {
  missing_count++;
  visible_count = 0;
}


// ============================================================
// 8.3 CHANGE THE BRIGHTNESS OF THE EYES.
//
if(ball_is_found) {
  int brightness = 0;

  //
  // Calculate brightness from ball_area.
  // (0 <= brightness < 255)
  //

  //
  // brightness = ???
  //

  setEyeLEDColor(brightness * 0x010101);
}
```

```
// ========================================================
// 8.4 TURN THE ROBOT EYES ON THE BALL.
//
if(ball_is_found) {
  // Track the ball. (the exercise 4)
  //
  // step_pan = ??
  // step_tilt = ??
  //
}

if(ball_is_found || (!ball_is_found && missing_count < 10 )) {
  // Update the target pan and tilt.
  pan = current_pan + step_pan;
  tilt = current_tilt + step_tilt;
}

// Movable Range of pan and tilt joints
pan = MAX(pan, pan_min);
pan = MIN(pan, pan_max);
tilt = MAX(tilt, tilt_min);
tilt = MIN(tilt, tilt_max);


// ========================================================
// 8.5 CHANGE THE TARGET OF THE JOINT ANGLES
// AND UPDATE THE MODE IF NECESSARY.
//
if(mode == MODE_WAIT) {
  //
  // WAIT (DEFEND)
  //

  // rpos0 = ??
  // rpos1 = ??
  // rpos2 = ??
  // lpos0 = ??
  // lpos1 = ??
  // lpos2 = ??

  if(ball_area > ball_area_threshold) {
    //
    // If the robot is waiting and
    // the ball with the sufficient area is found,
    // switch to the attack mode.
    //
    mode = MODE_ATTACK;
  }
}
```

```
else if(mode == MODE_ATTACK) {
  //
  // ATTACK
  //

  double margin = 0.1;
  // rpos0 = ??;
  // rpos1 = ??;
  // rpos2 = ??;
  // lpos0 = ??
  // lpos1 = ??
  // lpos2 = ??

  //
  // If the current joint angles are sufficiently near to
  // the target angles,
  // switch to the return mode.
  //
}
else if(mode == MODE_RETURN) {
  //
  // RETURN
  //

  double margin = 0.1;
  rpos0 = 0.0;
  rpos1 = 0.72;
  rpos2 = 0.9;
  // lpos0 = ??
  // lpos1 = ??
  // lpos2 = ??

  if(fabs(current_rpos0 - rpos0) < margin
     && fabs(current_rpos1 - rpos1) < margin
     && fabs(current_rpos2 - rpos2) < margin) {
    //
    // If the current joint angles are sufficiently near to
    // the target angles,
    // switch to the wait mode.
    //
    mode = MODE_WAIT;
  }
}


// ============================================================
// 8.6 SEND THE LATEST TARGET ANGLES TO MOTORS.
//

// Update the joint angles.
```

```
    setPosition(ShoulderL, lpos0);
    setPosition(ArmUpperL, lpos1);
    setPosition(ArmLowerL, lpos2);
    setPosition(ShoulderR, rpos0);
    setPosition(ArmUpperR, rpos1);
    setPosition(ArmLowerR, rpos2);

    setPosition(Neck,pan);
    setPosition(Head,tilt);

    /*
    cerr << role_name << ": "
         << "pan:" << rad2deg(pan) << ", tilt:" << rad2deg(tilt) << endl;
    cerr << role_name << ": "
         << "mode:" << mode << ", ball_area:" << ball_area << endl;
    */

    // step
    myStep();
  }
}
```

付録 1 ：プログラミングに必要な定義済み関数と定数値

ソースコード 8: 定義済み関数と定数値

```
// モータ制御
  void setPosition(int joint, double angle);
  // 各関節モーター（ID: joint）の角度値(angle)（単位radian）の目標値をセットする。

  double getPosition(int joint);
  // 各関節モーター（ID: joint）の現在の角度値(angle)（単位radian）を取得する。

  double getMinPosition(int joint);
  double getMaxPosition(int joint);
  // 各関節モーター（ID: joint）の角度値(angle)（単位radian）の
  // 最小値または最大値を取得する。

  // 各関節のID は下記の名前で参照できる。
  enum Joints {
     ShoulderR /*ID1 */, ShoulderL /*ID2 */, ArmUpperR /*ID3 */, ArmUpperL /*
         ID4 */,
     ArmLowerR /*ID5 */, ArmLowerL /*ID6 */, PelvYR /*ID7 */, PelvYL /*ID8 */,
     PelvR /*ID9 */, PelvL /*ID10*/, LegUpperR /*ID11*/, LegUpperL /*ID12*/,
     LegLowerR /*ID13*/, LegLowerL /*ID14*/, AnkleR /*ID15*/, AnkleL /*ID16*/,
     FootR /*ID17*/, FootL /*ID18*/, Neck /*ID19*/, Head /*ID20*/
  };

  void setControlP(int joint, double pgain);
```

```
  // 各関節モータ(joint)のP 制御ゲイン(pgain)をセットする。

  void setControlPAll(double pain);
  // すべての関節モータにP 制御ゲイン(pgain)をセットする。

  void setMotorForce(int joint, double force);
  // 各関節(joint) の力（トルク）の目標値をセットする。

  void enableArmMotorForceFeedback();
  // 腕の各関節の現在の力(トルク)を取得できるようにする
  //（デフォルトでは取得できない)。

  double getMotorForceFeedback(int joint);
  // 各関節(joint) の現在の力（トルク）を取得する。

  double deg2rad(double deg);
  // 角度値(degree)を角度値(radian)に変換する。

  double rad2deg(double rad);
  // 角度値(radian)を角度値(degree)に変換する。

// 歩行制御
  void walkStart();
  // 歩行制御開始を指示する。足踏みを開始する。

  void walkStop();
  // 歩行制御停止を指示する。足踏みをやめる。

  bool isWalking();
  // 歩行制御開始しているかどうかを返す。
  // 歩行制御中はtrue, そうでなければfalse を返す。


  // X 軸（ロボット前方）への移動バイアスを mag にセットする。
  // -1 <= mag <= 1の値をとり、正だと前進、負だと後退。

  void walkSetYAmplitude( double mag );
  // Y 軸（ロボット横方向）への移動バイアスを mag にセットする。
  // -1 <= mag <= 1の値をとり、正だと左へサイドステップ、負だと
  // 右へサイドステップ。

  void walkSetAAmplitude( double mag );
  // A 軸（ロボット体軸）中心の旋回バイアスを mag にセットする。
  // -1 <= mag <= 1の値をとり、正だと逆時計回り、負だと時計回りに旋回。

  void walkSetMoveAimOn( bool b );

// 登録動作リプレイ
  void playPage(int page);
  // page で指定される登録モーションを実行する。実行が終わるまで他の
```

```
  // 制御は止まる(同期実行)。
  // 主な登録モーションの動きは以下のとおり。
  // 0-255まであるが、新しい動作を登録するのはロボット実機が必要と
  // なりやや面倒なので、どうしても作りたい人がいれば申し出ること。
  enum Motions {
    Initial=1, WalkReady=9, UprightF=10, UprightB=11,
    RightKick=12, LeftKick=13, Hello=24
  };

// 画像処理
  void enableCamera();
  // 頭部カメラを利用できるようにする(デフォルトでは動作しない)。

  int getImageWidth();
  int getImageHeight();
  // カメラセンサから取得できる画像の幅と高さを取得する。

  char* getImage();
  // カメラから画像を取得する。RGB の順に並ぶ一次元配列。
  // 簡単な画像処理もできるが煩雑なので省略。処理を実装したい人は
  // 申し出ること。

  bool getRedBallCenter(double &x, double &y);
  bool getBlueBallCenter(double &x, double &y);
  bool getYellowBallCenter(double &x, double &y);
  // カメラから画像を取得してそれぞれ赤、青、黄色のボールの画像中の
  // 位置を返す。引数x,y は C++の参照渡しなので書き換わる。特定の色の
  // 画素の重心を求めている単純な処理。

  bool getRedCurtainCenter(double &x, double &y);
  bool getBlueCurtainCenter(double &x, double &y);
  bool getYellowCurtainCenter(double &x, double &y);
  // カメラから画像を取得してそれぞれ赤、青、黄色の領域が
  // 視界の 50%以上を占めていれば true を返す。
  // また、引数x,y には各色の領域の重心位置を返す。
  // 引数x,y は C++の参照渡しなので書き換わる。特定の色の
  // 画素の重心を求めている単純な処理。

// LED Control
  void setEyeLEDColor( int color );
  // 両目の位置にあるLED の発光色を指定する。RGB を上位 24 ビット目から
  // 8ビットずつ指定した整数で指定する。

  void setHeadLEDColor( int color );
  // 額のLED の発光色を指定する。

// キーボード入力
  int keyboardGetKey();
  // キーボードで押されているキーのASCII コード（文字コード）を取得する。
  // ただしscanf のように文字列や数値を入力することは出来ない。スペース
```

```
  // バーやA,B,C などのキーの判別ができる。switch-case 文などをつかって動
  // 作を切り替えることができる。
  // 矢印キーはKEYBOARD_UP, KEYBOARD_DOWN, KEYBOARD_LEFT,
  // KEYBOARD_RIGHT で定義されている。

// プログラム制御
  void run();
  // ロボットを制御する関数。課題ではこの中を書き換える。
  // while ループの中を繰り返し実行する。

  void myStep();
  // 時間を一単位進める。ループの最後には必ず入れておく。
  // 途中で時間を進めたいときにも使える(モーター、センサーの値などが
  // 変わる)。

  void wait(int ms);
  // 指定した時間(msec) 待つ。歩行制御は停止しない。

// メッセージ表示
  cout << "message" << endl;
  // 標準出力にメッセージを出力する。黒表示される。endl は改行。

  cerr << "(" << x << "," << y << ")";
  // 標準エラー出力に (x,y)を表示する。x と y の変数値が表示される。
```

付録２：USB メモリの使用方法

11.9　仮想マシンでの接続準備

仮想マシンで動作している Linux 環境に USB メモリを認識させるにはホスト OS(Windows) から USB メモリを切断して、仮想マシンに接続する必要がある。仮想マシンに接続するとホスト OS の管理から離れるため “デバイスは安全に取り外せます” といったメッセージが表示されるが仮想マシンに接続している間は取り外さないように注意すること。また、USB メモリを取り外す際はまず 11.11 の手順で Linux 環境から取り外した上で 11.12 に従い、仮想マシンへの接続を解除しなければならない。

USB メモリを仮想マシンに接続する手順は以下の通り。

1. USB メモリを PC に接続する。
2. VMware Player のデバイス設定アイコン (図 44) を右クリックしてメニューを表示する。。アイコンが非表示状態の場合は図 45 の “《” 部分をクリックしてアイコンを表示させてから上記を行う。
3. 表示されたメニューの “接続 (ホストから切断)” をクリックする (図 46)。
4. ホスト OS から切断される旨の注意が表示されるので “OK” をクリックする (図 47)。
5. 仮想マシンに接続されると、Linux 環境側で自動的に認識される。Linux 環境での USB メモリの利用法は 11.10 を参照すること。

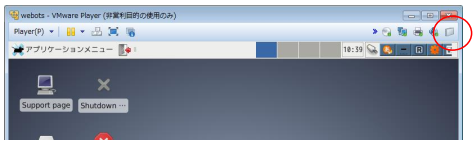


図 44: デバイス設定アイコン

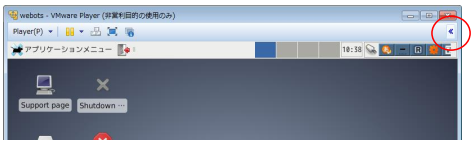


図 45: アイコンが非表示の状態

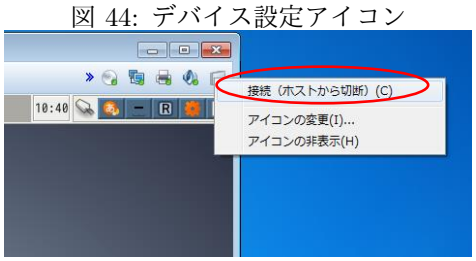


図 46: デバイス設定メニュー

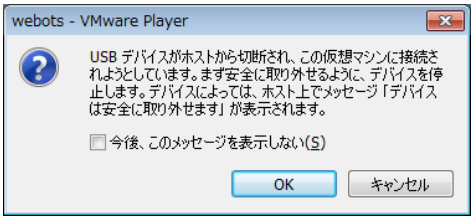


図 47: 仮想マシン接続時のダイアログ

11.10 Linux での接続

Live DVD 環境では USB メモリを接続すると、自動的に**マウント**され、使用可能となる (図 48)。

デスクトップには接続した USB メモリのアイコンが追加され、ここから USB メモリ内のファイルを表示することもできる (図 49)。

11.11 Linux での取り外し

USB メモリを取り外す際には、USB メモリへの書き込みを完了させるために**アンマウント**の手続きが必要となる。

デスクトップ上の USB メモリのアイコンを右クリックすると表示されるメニューの**取り出し**をクリック (図 50) すれば USB メモリが**アンマウント**され、取り外す準備が完了する。デスクトップからもアイコンが消えていることを確認し、USB メモリを取り外す。

11.12 仮想マシンからの接続解除

USB メモリの仮想マシンへの接続を解除するには、まず 11.11 の手順で USB メモリを Linux 環境から取り外す必要がある。次に VMware Player のデバイス設定アイコンを右クリックし、メニューの “切断 (ホストに接続)” をクリック (図 51) すれば仮想マシンへの接続が解除されてホスト OS(Windows) から参照できるようになる。PC から取り外す際の手続きは通常の USB メモリと同様である。

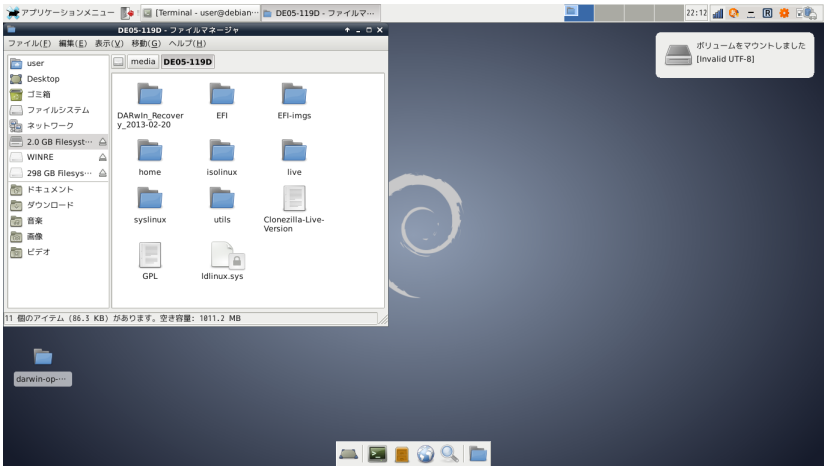


図 48: USB メモリを接続すると自動的にマウントされる

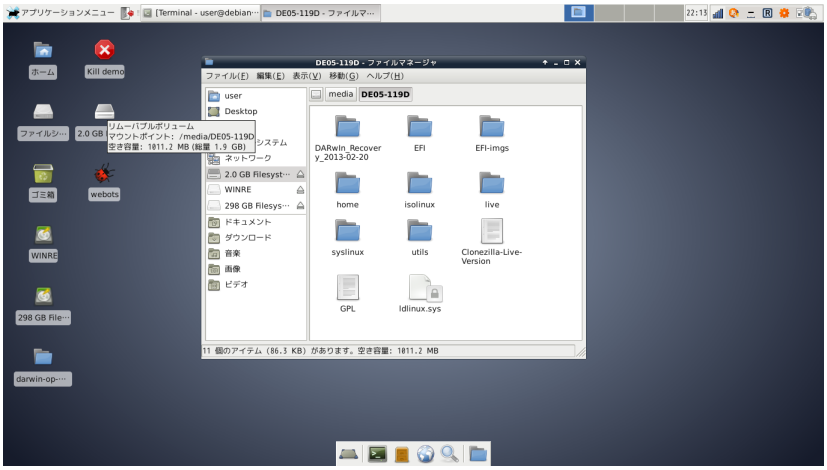


図 49: デスクトップに USB メモリのアイコンが追加される

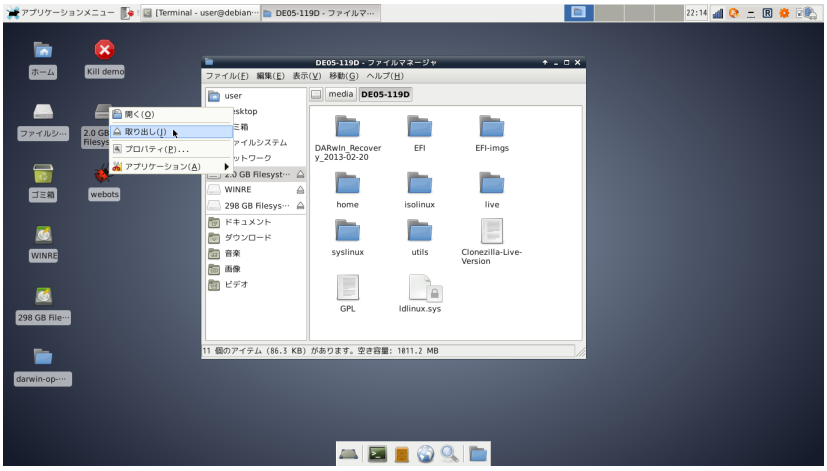


図 50: アンマウント

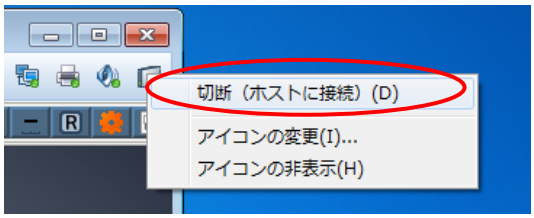


図 51: 仮想マシンへの接続の解除