# Roboko: ソースコードに写真を貼り込める統合開発環境

加藤 淳　　坂本 大介　　五十嵐 健夫*

**概要.** プログラムのソースコードは，大抵の場合，文字や記号表現を用いて記述するものである．このような表現は，論理的な処理の精確な記述に適している一方で，画像や多関節ロボットの姿勢情報など，具体的なデータセットを分かりやすく見せることができない．この問題は，実世界の人やロボットの姿勢情報を扱うプログラミングで顕著である．そこで我々は，人やロボットの姿勢データを写真と紐づけ，ソースコード中に写真を貼り込める開発環境 Roboko を試作した．まず，写真と紐づけられた姿勢データを第一級オブジェクトとして扱う拡張を既存の文字言語に施した．次に，姿勢データを引数に取る API ライブラリを提供した．そして，エディタに写真を貼り付けられるような開発環境を実装した．本稿では，関連研究を示し，試作した開発環境の使い方と実装の概要を紹介して，今後の課題を議論する．

## 1　はじめに

プログラミング言語は，プログラマがコンピュータに実行してほしい処理を記述するためのインタフェースである．したがって，プログラミング言語は，コンピュータにとって精確な解釈が可能なように構文や文法が定められているべきであるのと同時に，プログラマにとっても容易に理解でき，使いやすくあるべきである．そこで，開発環境の使い勝手を向上してプログラミング体験をよりよいものにする試みがなされてきた．例えば，構文要素として文字の代わりに記号表現を用いるフローチャートや，構文エラーを起こさないようにする構造化エディタ，コンテキストに応じて入力可能な内容が視覚的に表示されるコード補完の改良 [3] などが挙げられる．しかしながら，プログラミング言語自体は文字や記号的な表現を用いるものがほとんどである．このような表現は個々の構文要素に対して明快な意味づけができ，論理的な処理の精確な記述に適している一方で，画像や多関節ロボットの姿勢情報など，具体的なデータセットを分かりやすく見せることができない．ソースコードには，ほとんど無意味な数値か，別途ファイルやデータベースに保存されたデータを参照するための定数や文字列のキーが並ぶことになる．

この問題に対して，文字だけでなくマルチメディアコンテンツを貼り込めるソースコードエディタを実装するためのフレームワーク [2] が提案されており，画像処理を行うメソッドのコメントに画像を貼り込んで処理内容を分かりやすく表す例が紹介されている．具体的なデータセットについても，文字より分かりやすいマルチメディア表現があれば意味が読み取りやすくなるのは明らかだろう．実際に，画像データをそのまま画像としてプログラムの中に貼り込める Sikuli[7] という開発環境が提案されている．この開発環境を用いると，デスクトップ上に現れる特定の要素を探したり，その要素が見つかった場合にクリックするなどの作業を自動化する GUI 用のマクロを直感的に書いたりできる．

我々は，実世界における人やロボットの姿勢情報を扱うプログラミングで同様の問題が生じることに着目した．例えば，モーションキャプチャシステムや Kinect を用いれば人の姿勢が得られ，各関節にサーボモータを備えたロボットであれば現在の姿勢を得られるほか，望みの姿勢を取らせることもできる．しかし，姿勢情報そのものは数値の羅列であり，人がそのまま理解するのは難しい．そこで，各姿勢情報を可視化し，ソースコード中に視覚表現として貼り込めるようにすることで解決を図った．

前出の Sikuli ではキャプチャされる情報が画像であり，そのままプログラマにとって分かりやすい表現であったのに対し，我々の問題設定でキャプチャされる姿勢情報は簡単に可視化できるものではない．また，人やロボットの姿勢は常に実世界における文脈の中に置かれており，姿勢情報をボーンとして可視化するだけでは情報量が不足していると考えられた．そこで，カメラを用いてそのときの写真を撮影することとし，プログラマが状況を想像・理解しやすくした．すなわち，本研究の新規性は，プログラムの実行には不要である冗長な情報 (写真) を自動的に取得し，文字ベースの開発環境に統合することで，プログラマのユーザビリティを向上した点にある．

本研究では，姿勢情報を用いるプログラミングのワークフローを支援するため，既存の文字言語に拡張を施し，ライブラリと開発環境を実装した．まず，Processing[1] 言語に姿勢情報を第一級オブジェクトとして扱えるよう拡張を施した．また，姿勢情報



* Jun Kato, 東京大学 / 日本学術振興会, Daisuke Sakamoto and Takeo Igarashi, 東京大学 / JST ERATO 五十嵐デザインインタフェースプロジェクト

[1] Processing. http://processing.org/

を引数に取る API ライブラリを提供した．そして，写真を貼りこめるよう拡張したエディタを含む文字ベースの統合開発環境に，姿勢情報と写真の記録および紐づけを自然に行える例示インタフェースと，記録した写真を閲覧できる “Pose library” を追加した．人の姿勢取得には Kinect[2] を用い，ロボットは LEGO Mindstorms NXT[3] で作られたものに対応した．プログラマは，例えば “Pose library” に記録された人やロボットの姿勢を現在の状態と比較したり，ロボットに指定した姿勢を取らせたりできる．そして，これらの機能と Processing の GUI を始めとする様々なライブラリを組み合わせ，インタラクティブなアプリケーションを開発できる．

以降，関連研究を示し，試作した開発環境の使い方と実装の概要を紹介した上で，姿勢情報の可視化手法として写真を使うことの意義を論じ，今後の課題を述べる．

## 2 関連研究

本研究は，実世界における人やロボットの姿勢情報を扱うインタラクティブなアプリケーションの開発を支援するものである．2010 年の Kinect 発売以来，人の姿勢情報をリアルタイムに取得してインタラクティブなアプリケーションを開発することが容易になってきた．そして，アプリケーション開発を支援するため，ユーザの姿勢を判定したり，ジェスチャを認識したりできる多くのツールやライブラリが提案されてきた．しかしながら，これらのツールやライブラリは，プリセットの姿勢やジェスチャを使わない場合，開発環境とは独立したツールを用いて姿勢やジェスチャを学習させる過程が必要である．これに対し，本研究は，姿勢情報を扱うアプリケーションの開発過程全体を，ライブラリと開発環境の密結合で支援する．このように特定分野のアプリケーションの開発過程を支援する研究は，例えば新しいデバイスのプロトタイピング [1] や機械学習 [4] について行われてきている．

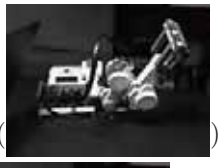
ロボットの姿勢情報を扱うアプリケーションの開発は古くから行われており，Microsoft Robotics Developer Studio[4] や LEGO Mindstorms のような文字や記号ベースの統合開発環境か，Choreonoid[11] や RoboJockey[6] のようなある姿勢から別の姿勢への遷移の連続をデザインできるエディタを用いるのが一般的である．前者はどのような姿勢を扱っているのかソースコードから想像がつきにくい．これは，前出の人の姿勢情報を利用するアプリケーション開発でも同様に表れる問題である．後者のようなエディタは，ユーザの入力に対して反応するような制御構造を必要とするインタラクティブなプログラムを組めないという欠点がある．そこで本研究では，人やロボットの姿勢情報を写真と紐づけ，写真をソースコード中に貼り込めるようにすることで，それぞれの利点を併せ持つ開発環境の実現を目指した．

なお，具体的なデータを活用するプログラミング手法として，例からのプログラミング (Programming by Example, PBE) がある．PBE では，ユーザはプログラムに望む入力と出力の具体例をシステムに与え，システムはその例を制約として満たすプログラムを生成する．いわゆる言語の文法を覚える必要がないため，プログラミング初心者や子供向けの応用が多くある．例えば，絵を描いて，ある状況から別の状況への置換ルールを指定してアニメーションやゲームを実装できる Viscuit[9] や，ロボットの各関節について動きを記録して再生できる Topobo[5] が挙げられる．これらのシステムで作成できるアプリケーションは，論理的抽象的な処理を自由に記述できる文字や記号ベースの開発環境と，具体例一つだけを作成するエディタの，中間の自由度を持っていると言える．一方で，本研究では，文字や記号ベースの開発環境が持っている自由度を保ったまま利便性を向上しようとする試みである．

## 3 Roboko によるプログラミング

本節では Roboko によるプログラミング体験を，以下のソースコードを開発する過程として紹介する．プログラミングは，人やロボットの姿勢情報を写真とともにキャプチャする例示から始まる．そして，写真をエディタにドラッグ&ドロップしたり文字を入力したりすることでソースコードを書き上げ，プログラムを実行する．ソースコードはキャプチャした写真と姿勢情報のデータセットと共に再配布できる．

**ソースコード 1．人が手をあげたらロボットが手を振る**

```
1 if (human.getPose().eq(  , 0.04)
2     && !robot.isActing()) {
3     if (flag) { robot.setPose(  ); }
4     else { robot.setPose(  ); }
5     flag = !flag;
6 } // 注: 変数の宣言と初期化などは省略した．
```

[2] Kinect. http://kinectforwindows.com/
[3] LEGO Mindstorms. http://mindstorms.lego.com/
[4] MRDS. http://www.microsoft.com/robotics/

### 3.1 姿勢情報のキャプチャ

プログラマは，Robokoのメイン画面にある “Pose library” の “+” ボタンをクリックしてプレビュー画面を開き，カメラの映像と姿勢情報をリアルタイムに確認できる．プレビュー画面では姿勢情報の入力ソースをKinectとロボットから選択できる．Kinectはコンピュータに接続されている1台が自動的に認識される．ロボットは，初めて開発環境に繋ぐ際に，同画面からBluetoothアドレスを入力する．写真を撮影するためのカメラは，Kinectが接続されている場合はそのRGB画像ストリームが用いられるが，別途Webカメラを接続して使うこともできる．

プレビュー画面には “Capture”ボタンがあり，カメラの前でロボットに望みの姿勢を取らせた状態でクリックすると，写真と姿勢情報のデータセットが新しく “Pose library” に登録される．ロボットの各関節を構成するサーボモータは，プレビューの最中現在の角度を維持するようなトルクがかかっているが，人が力をかけて角度を変え始めると自由回転するようトルクが解除される．これにより，ロボットの各関節を一つずつ動かしてポージングを行える．人の姿勢をキャプチャする場合にはプログラマ自身がポージングを行うことが想定されるので，Kinectの音声認識エンジンを利用し，“capture” と発話することでも新しいデータセットが登録されるようになっている．各データセットには “New pose (1)” のような一意の名前が自動的に付けられ，プログラマが後から自由に変更できる．


**図 1. Robokoのメイン画面とプレビュー画面.**

なお，プレビュー画面表示中は，“Pose library” は接続したロボットの姿勢情報を表す写真のみ選択可能になる．プログラマは，その中から望みの姿勢を選び，右クリックメニューから姿勢を復元できる．このように，既存のロボットの姿勢を元に改変を加え，新しい姿勢をキャプチャすることが容易な設計となっている．本機能を始め，ロボットの姿勢情報を取得する手法に関しては [8] を参考にした．

### 3.2 写真を用いたコーディング

プログラマは，Processingと互換性のある言語とライブラリを使ってコーディングを行える．Processingとの機能上の差異は，特定のAPIの引数に写真を使えることだけである．写真は，“Pose library” から選んでソースコードエディタ中のAPIの引数部分にドラッグ&ドロップできる．現在，以下のようなAPIがサポートされている．

**表 1. 写真を引数に取るAPIの利用例一覧.**

| |
|---|
| $robot.getPose().eq($  $, 0.04)$<br>姿勢の比較 (第2引数は誤差の許容範囲 $0 \sim 1$) |
| $robot.setPose($  $)$<br>特定の姿勢を取らせる (ロボットにのみ使用可) |
| $Action\ a = robot.action();$<br>$a = a.pose($  $).wait();$<br>$a = a.pose($  $);$<br>$a.play();$<br>連続した姿勢の変更を定義して実行する |

### 3.3 プログラムの実行

プログラマは，プレビュー画面が表示されていなければ，いつでもメイン画面の “Run”をクリックしてプログラムを実行できる．コンパイル時，またはプログラム実行中にエラーが起きたら，メイン画面

のステータスバーに内容が表示される．プログラマは，これまでに述べた姿勢情報のキャプチャ，コーディング，実行の過程を自由に行き来して，アプリケーションを作成できる．アプリケーションが完成したら，ソースコードおよび参照されている写真と姿勢情報のデータセットを含むZIP形式の書庫を作成して，他のマシン上で実行することもできる．

## 4 実装

### 4.1 概要

Robokoは，オープンソースプロジェクトであるProcessingの統合開発環境のユーザインタフェース部分を一から書き直して実装されている．その他の目に見えない変更点は，コンパイルの過程で写真を引数に取るAPIを実現する拡張ライブラリに自動的にリンクするようにしたことと，アプリケーションの実行を監視し，その間は開発環境側でKinectやロボットに接続しないようにしたことのみである．したがって，シンプルな言語仕様や，Javaベースのライブラリを利用できる拡張性は，オリジナルのProcessingと同等となっている．ロボットの姿勢の取得と制御，Webカメラによる画像の撮影機能はJava用のツールキットMatereal[10]を用いて実装した．人の姿勢の取得および画像の撮影にはKinect for Windows SDKを用いたが，これはJavaに対応していないため，必要に応じてWindowsのネイティブプロセスとして起動し，開発環境とTCP/IPで通信する別プログラムとして実装した．

### 4.2 ソースコードエディタ

ソースコードエディタに貼り付けられる写真は，Roboko内部では特定のAPIへの呼び出し文字列（$Pose.load\,(pose\_key)$）に置換され，保持されている．$pose_key$は3.1節で指定した一意の名前である．“Pose library”とエディタ間のドラッグ&ドロップでやり取りされるのも，コンパイラに渡されるのもこの文字列表現である．ソースコードの文字列表現は，エディタへ入力がある度にProcessing言語のパーサに渡され，抽象構文木が作られる．この抽象構文木を元に，エディタ上で構文要素が強調表示され，特定のAPIへの呼び出しと解釈された部分が写真に置換される．このようにして，ソースコードの文字列表現とエディタ上での見た目が同期するようになっている．文字列とその他のマルチメディアが混在したソースコードエディタの実装の詳細については[2]に詳しい．ただし，当時よりもテキストエディタを作るための標準ライブラリが進歩しており，例えば抽象構文木の個々の要素にテキストボックスを一つずつ割り当てるような力業はもはや必要なくなっている．近年の例だと，オープンソースで公開されている[7]が参考になるだろう．

### 4.3 API

本小節ではRobokoが提供するProcessingの拡張ライブラリに含まれる主要なクラスについて簡単に説明する．姿勢を取得したり制御するための処理は，現時点では$KinectHuman$と$Robot$クラスに書かれている．$Robot$クラスはインスタンス化するときにロボットの種類を文字列で指定する．そして，この種類名に応じて，利用されるMatereal[10]のロボットクラスと姿勢制御用のクラスが選択される．こうして，今後対応する姿勢の取得手法やロボットの種類を増やせるよう，十分に拡張性のある構成にしている．

姿勢情報は$Pose$クラスのインスタンスとして表され，人やロボットの種類ごとに子クラスが用意されている．姿勢情報と写真はそれぞれ一意の名前(例:$Hand\,up$)を持つテキストファイル($Hand\,up.txt$)とJPEG画像($Hand\,up.jpg$)として同一ディレクトリに保存され，テキストファイルは，姿勢情報のクラス名に続いて，姿勢を表す数値の羅列が続く簡単な仕様になっている．姿勢同士の比較は同じ種類でのみ可能(人同士，Mindstorms NXTの同じロボット同士)で，各関節角同士の差の絶対値からなるベクトル(肘の角度の差の絶対値，膝の角度の差の絶対値…)のユークリッド距離を0～1に正規化した値が閾値以下なら同じと判断する．

## 5 写真を撮ることに関する議論

本研究で扱っている人の姿勢情報は，実は，それ単体で可視化できる．どの関節がどれだけの角度かということが分かっているからだ．また，ロボットに関してもパーツの接続関係が既知であれば可視化できる．しかし我々は，あえて写真を撮ることにした．この選択にはいくつか理由がある．

初めに，最大の理由は，写真に勝手に含まれる環境情報である．例えば，姿勢情報には，人やロボットがいったい何をしている姿勢なのかという情報が含まれていない．人であれば，コップを持ってジュースを飲もうとしているときと，望遠鏡を片手に天体観測をしようとしているときで，同じような姿勢になるかもしれない．ロボットの手であれば，小さなボールと大きなボールをつかもうとするとき，手の広がりようが多少違うだけで，やはり同じような姿勢になるだろう．これらの区別は，数値で見れば傾向としてつかめるかもしれない．しかし，3D CGでボーンを可視化した場合にほとんど差が分からない可能性が高い．さらに，数値上で差が出なくても，プログラムが動作するための暗黙の前提条件が写っていることがある．例えば，ロボットの動作する床面がカーペットだったのかフローリングだったのかによって，ロボットの動きは大きく左右されるだろう．自分が一年前に書いたソースコードを見直した

り，他の人が書いたソースコードを読むとき，どのようなハードウェアのセットアップでプログラムが実行されていたのかが一目で分かることは非常に重要である．

次に，二つ目の理由として，ロボットに関してパーツの接続関係を一々入力する手間を省きたかったことが挙げられる．一般的なロボットの開発環境でそのようなトポロジ情報が必要となるのは，ロボットの姿の CG による可視化のみならず，ロボットが転倒しないような姿勢を計算するため，物理シミュレータの動作に使われるからである．しかし，我々のユースケースでは，写真が撮れたということは，与えられた姿勢情報で実物のロボットが安定して自立したということを意味している．このように，写真とともにキャプチャされた姿勢情報を信頼することとし，シミュレータを内蔵しなかった．ロボットがある姿勢から別の姿勢への遷移の過程で加速度がついて不安定になり，転倒するケースも考えられるが，これについてはプログラマの試行錯誤で解決されることを期待している．我々の開発環境のベースとなった Processing は，ダウンロードしてすぐに起動でき，ソースコードを書いて実行ボタンをクリックすればプログラムが起動するシンプルなプログラミング体験で人気を不動のものとした．我々もこのミニマリズムを継承している．

さらに，我々は，写真を撮るという行為をプログラミング体験に埋め込むことで，プログラマとユーザの距離が縮む可能性に期待している．姿勢情報を扱うプログラミングでは，個人差や個体差が非常に大きい．例えば，プログラマが自分の身体で試してチューニングしたプログラムが，テストユーザに対してはうまく動かないことが多々ある．我々は，このような場合に，テストユーザの姿勢を新たにキャプチャしてプログラマの姿勢のデータセットと入れ替えたりする，非常にカジュアルなプログラミングを想定している．結果として，プログラマの姿勢データセットしか含まれていなかった “Pose library” にはテストユーザの写真が増えていき，ソースコードには何人もの写真が登場することになるだろう．こうして，プログラマだけのものだったソースコードが，多くのユーザと共有できるものとなる．

## 6 今後の課題

我々は，5 節の議論を検証するユーザスタディが必要であると考えている．また，API の充実と，写真ではなく動画を使ったインタラクション，姿勢情報以外への応用も視野に入れている．

### 6.1 API の充実

本研究では，姿勢情報を第一級オブジェクトとして扱うため，その柔軟で高度な演算を可能にする API と，API への引数としてオブジェクトを構築できる便利なユーザインタフェースを提供することは重要な課題である．

現在の Roboko は，特定の姿勢情報と現在の状況を比較することしかできない．例えば，右手を高く上げている状態と，少し低く上げている状態，両方を同じ “右手を上げている状態” として検出したいと思ったら，2 つの if 文を書くことになる．そして，それぞれについて検出の閾値を試行錯誤して得なければならない．この問題は，複数の姿勢情報を機械学習させる PBE によって比較的簡単に解決できるだろう．例えば，“Pose library” から複数の写真を選択して “Pose set” を定義できるようにし，このセットに含まれる姿勢情報を正例，それ以外を反例とするようなインタラクションが考えられる．

また，現在の姿勢比較は，全身同士しか比較できない．例えば先の例で左手は上げていても下げていてもよい，というようなユースケースが考えられる．Kinect を利用して写真を撮っている場合は，画像中の各関節の位置が分かっているため，例えばソースコードエディタに貼りつけた写真のうち無視したい関節部分をマスクで塗りつぶすことで，その部分は比較に使わないようにする，といったインタラクションが可能だろう．Kinect で撮影された写真ではユーザが概ねカメラに正対しており，部分選択が容易なケースが多いと考えられるが，被り物をしたロボットなど，もし難しい場合は，関節情報を 3D でレンダリングし，回転させられるようにするなど既存手法と相補的に併用する必要があるだろう．

### 6.2 動画への対応

我々は，Roboko のソースコードエディタに，写真だけでなく動画を貼り込めるようにする予定である．写真は，姿勢情報のある一瞬を切り取ったものである．そのため，人のジェスチャを認識したり，ロボットに時間のかかる動きをさせようとすると，一連の動作を何枚かのキーフレームとなる写真が連なったものとして考え，プログラムを書くことになる．しかし，より直感的には写真ではなく動画を撮影し，それをもとにジェスチャ認識機を作れたり，ロボットの動きを再生したりできるのが望ましいように思われる．動画には再生速度というパラメータがある．ロボットにその動画と同じ動作をさせる際，動画の再生速度を調整することで，ロボットの動作速度まで変えられるようなインタラクションが考えられる．

### 6.3 Physical Computing 向け応用

Processing がマイコンのコンパイラを積んで Arduino[5] という開発環境に進化したように，Roboko のユーザインタフェースもまた，Arduino のようなハードウェアのプロトタイピング用開発環境として利用できると考えている．この場合，写真は配線の情

[5] Arduino. http://arduino.cc/

報を教示する役割を果たせるが，その裏にあるべき文字列表現は自明ではない．さまざまな可能性が考えられるが，例えばハードウェアのモジュール (Arduino 用シールドなど) 作成者がモジュールの写真をライブラリと一緒に配布し，利用者のプログラマは *#include* 文を書く代わりにモジュールの写真をエディタにドラッグ&ドロップできるかもしれない．

## 7 結論

本稿では，人やロボットの姿勢データを写真と紐づけ，ソースコード中に写真を貼り込める開発環境 Roboko を試作した．姿勢データとは，Kinect で得た人の関節角や，ロボットのサーボモータから得た回転角の情報を指す．写真と姿勢データの対応付けは，姿勢データの取り込み時にプログラマに対象の写真を撮影させることで自然に得ることができた．

## 謝辞

本研究は第 7 回 Microsoft Research CORE プログラムの支援を受けた．ここに感謝の意を表す．

## 参考文献

[1] B. Hartmann, S. R. Klemmer, M. Bernstein, L. Abdulla, B. Burr, A. Robinson-Mosher, and J. Gee. Reflective physical prototyping through integrated design, test, and analysis. In *Proc. of UIST '06*, pp. 299–308. ACM, 2006.

[2] A. J. Ko and B. A. Myers. Barista: An implementation framework for enabling new tools, interaction techniques and views in code editors. In *Proc. of CHI '06*, pp. 387–396. ACM, 2006.

[3] C. Omar, Y. Yoon, T. D. LaToza, and B. A. Myers. Active code completion. In *Proc. of ICSE '12*, pp. 859–869. IEEE Press, 2012.

[4] K. Patel, N. Bancroft, S. M. Drucker, J. Fogarty, A. J. Ko, and J. Landay. Gestalt: integrated support for implementation and analysis in machine learning. In *Proc. of UIST '10*, pp. 37–46. ACM, 2010.

[5] H. S. Raffle, A. J. Parkes, and H. Ishii. Topobo: a constructive assembly system with kinetic memory. In *Proc. of CHI '04*, pp. 647–654. ACM, 2004.

[6] T. Shirokura, D. Sakamoto, Y. Sugiura, T. Ono, M. Inami, and T. Igarashi. RoboJockey: real-time, simultaneous, and continuous creation of robot actions for everyone. In *Proc. of ACE '10*, pp. 53–56. ACM, 2010.

[7] T. Yeh, T.-H. Chang, and R. C. Miller. Sikuli: using GUI screenshots for search and automation. In *Proc. of UIST '09*, pp. 183–192. ACM, 2009.

[8] W. Yoshizaki, Y. Sugiura, A. C. Chiou, S. Hashimoto, M. Inami, T. Igarashi, Y. Akazawa, K. Kawachi, S. Kagami, and M. Mochimaru. An actuated physical puppet as an input device for controlling a digital manikin. In *Proc. of CHI '11*, pp. 637–646. ACM, 2011.

[9] 原田 康徳, 加藤 美由紀, R. Potter. Viscuit: 柔軟な動作をするビジュアル言語. In *Proc. of WISS '03*, pp. 41–48, Dec 2003.

[10] 加藤 淳, 坂本 大介, 五十嵐 健夫. matereal: インタラクティブなロボットアプリケーションのプロトタイピング用ツールキット. In *Proc. of WISS '10*, pp. 83–88, Dec 2010.

[11] 中岡 慎一郎, 三浦 郁奈子, 森澤 光晴. ヒューマノイドロボットのコンテンツ技術化に向けて：クリエイターによる多様な表現の創出が可能な二足歩行ヒューマノイドロボットの実現. *Synthesiology*, 4(2):80–91, 2011-05.

### 未来ビジョン

“Programmers are people, too.” という言葉がある．この，プログラマだって人である，という言葉には，プログラマがいかに分かりづらい言語を理解し，使いづらい開発環境と格闘してきたか，プログラマと一般人の溝がいかに深いか，という現状認識が現れているように思われる．

しかし私は，プログラマと一般人の溝が近年急速に埋まりつつあるように感じている．まず，開発環境を整えるのが簡単になった．以前なら必要なファイルを揃えてパスを通して，と大変な手間だったものが，今はブラウザを開くだけでよい．プログラミングの学習も Kahn Academy などでインタラクティブに進められる．また，プログラミングできる対象が身近になってきた．LEGO Mindstorms NXT のように実世界で動くものはもちろん，例えばスマートフォン上でスマートフォン用のプログラムを書いたり (TouchDevelop)，Web サービス同士を if-then ルールで連携させたり (IFTTT) できる．家電製品がネットワークに繋がるようになってきたから，これらを遠隔制御する実用的なプログラムも簡単に書けるようになるだろう．

また，プログラミング言語の研究者コミュニティでは，言語だけでなく開発環境まで含めたプログラマの体験を重視する研究が増えてきている．我々のコミュニティで道具としての UI から体験としての HCI に研究の主流がシフトしたように，言語，ライブラリ，開発環境を個々に評価するのではなく，全体のワークフローを重視する流れが生まれている．

我々は，このような追い風を受けながら，“People are programmers.” の世界を実現していきたいと考えている．