Jst Erato Asada Project

# A highly modular software toolset for robot motion development


Fransiska Basoeki [*] Fabio DallaLibera [†] Takashi Minato [‡§] Hiroshi Ishiguro [*‡]

[*] Graduate School of Engineering Science, Osaka University, Osaka, Japan
[†] Research Fellow of the Japan Society for the Promotion of Science, Osaka University, Osaka, Japan
[‡] Hiroshi Ishiguro Laboratory, Advanced Telecommunications Research Institute International (ATR), Japan
[§] ERATO, Japan Science and Technology Agency, Osaka, Japan

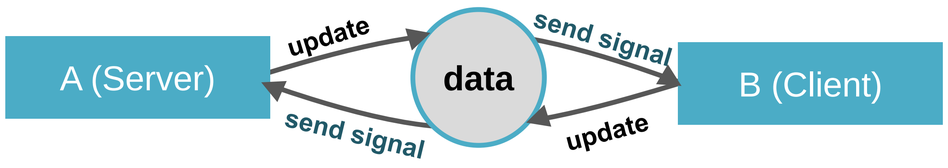


**Abstract**

We present the design and development of a software toolset for developing robot motions. The modularity of the software allows reusing the same code for different robots. The software can be reconfigured by loading XML files which contained information of the robot hardware. Users can easily monitor the status of the robot hardware, such as motors' connections and temperatures. Additionally, it features easy motion timeline editing and collision detection when a robot model is provided.


## Helium

Sharing **multiple variables** between **multiple programs** can be done easily. Conventional method requires the user to include TCP/IP programs and define their own command parsing. By using ***Helium***, the user no longer needs to write such code.




## Basic Class

```
template <GT,GP,
          ST,SR,
          UT,UP,
          bool getRequiresParam,
          bool setRequiresParam,
          bool connectRequiresParam>
class TypedCell {
  virtual void getData(GT &t)=0;
  virtual void getData(GT &t, const GP &p)=0;
  virtual void setData(GT &t)=0;
  virtual void setData(SR &t, const ST &p)=0;
  virtual void connect(CellConnection &c)=0;
  virtual void connect(CellConnection &c, const UP &p)=0;
  virtual void disconnect(CellConnection &c)=0;
};
```

when getData is called:
**GT:** type of return value
**GP:** type of parameter

when setData is called:
**ST:** type of the data to be set
**SR:** type of the return value

getRequiresParam
setRequiresParam
connectRequiresParam
} True when the function always require a parameter when called

in connect/disconnect function:
**UT** type of the connected data
**UP** type of the parameter needed

*Example:* Temperature of the robot can only be read (*double* type), cannot be set, and when it is updated, it will send a *double* value

```
class Temperature:
public TypedCell<double, void,
                 void, void,
                 double, void,
                 false,false,false>{
  virtual void getData(GT &t){...}
  virtual void connect(CellConnection &c){...}
};
```

```
struct DefaultSignature:public CellSignature{
  typedef PGT GT;
  typedef Void GP;
  typedef PGT ST;
  typedef StdReply SP;
  typedef PGT UT;
  typedef Void UP;
  static const bool getRequiresParam=false;
  static const bool setRequiresParam=false;
  static const bool connectRequiresParam=false;
};
```

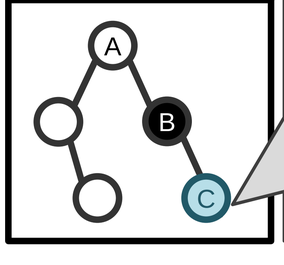
The 9 parameters above can also be passed as a struct (*CellSignature*), here is shown the default *CellSignature*, which is used when only GT is passed into *TypedCell*

```
class Position: public TypedCell<double>;
```

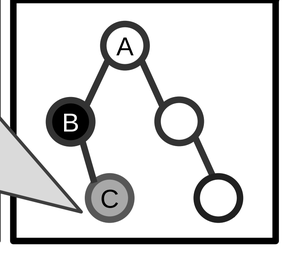
*double* is used as GT, ST, and UT

## Tree

### Server Side




### Client Side

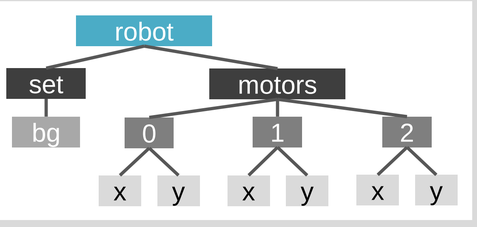



Data variables in ***Helium*** are represented as tree structures. Data to be shared between 2 programs are represented as *Cell* in the server side and *ValueProxy* in the client side. Both shares the same *CellSignature*, and are located on the same structure path even if each tree structure is different.

*Example:* Server and Client Side are sharing ***C*** which is located on path ***A-B-C***.

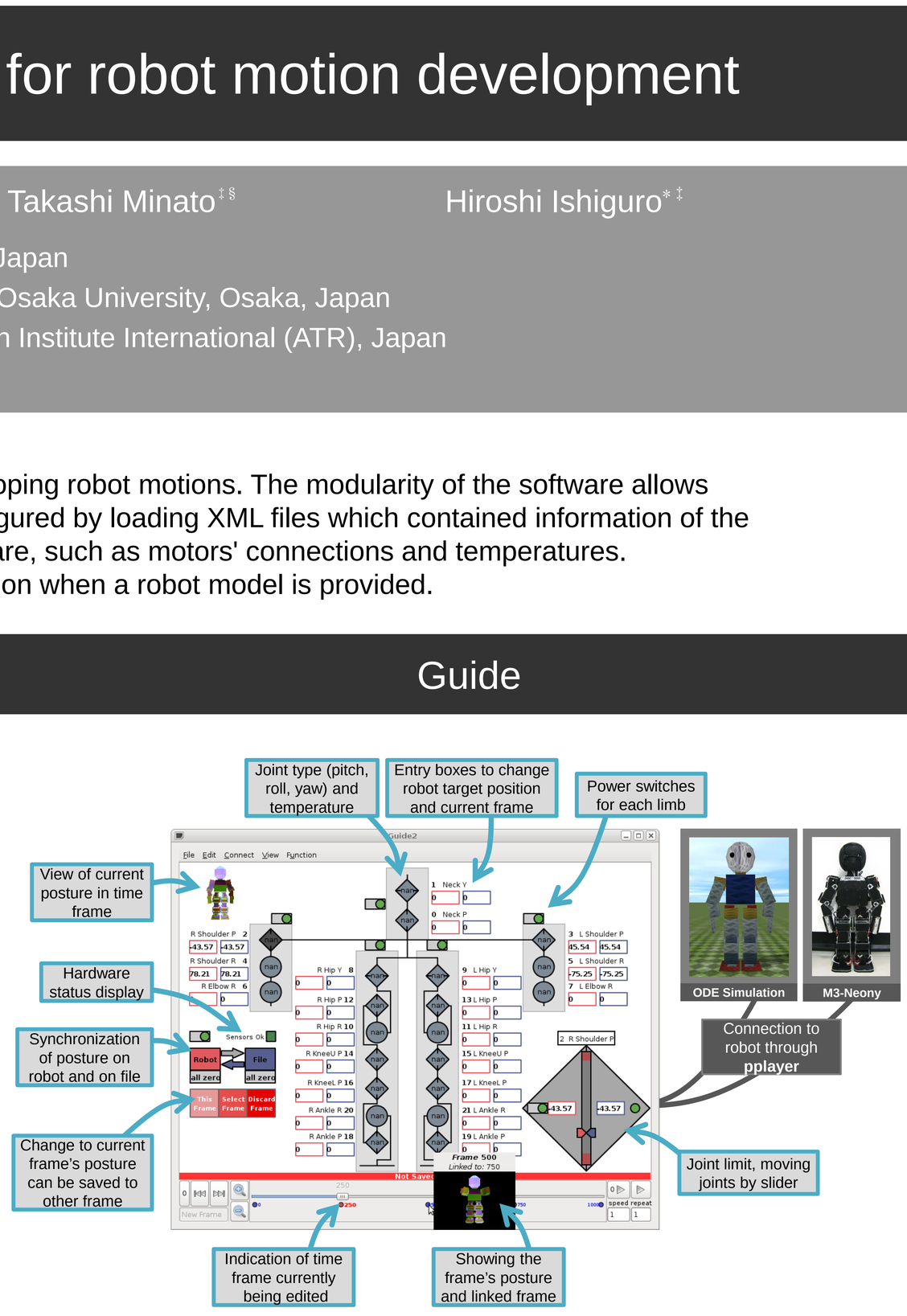
```
class Motor {
  int x;
  int y;
};

class Settings {
  std::string bg;
};

class Robot {
  Settings set;
  std::vector<Motor> motors;
};
```

```
template<> class DefDescr<Motor>:public Description<Motor>{
public:
  DefDescr(Motor& m):Description<Motor>(m){}
  void getMapMembers(GenericIDPath& hopname,std::vector<NodeMapping*> &members){
    members.push_back(scalarMapping(d(obj.x),IDPath("x",".")));
    members.push_back(scalarMapping(d(obj.y),IDPath("y",".")));
  }
};

template<> class DefDescr<Settings>:public Description<Settings>{
public:
  DefDescr(Settings& s):Description<Settings>(s){}
  void getMapMembers(GenericIDPath& hopname,std::vector<NodeMapping*> &members){
    members.push_back(scalarMapping(d(obj.bg),IDPath("bg",".")));
  }
};

template<> class DefDescr<Robot>:public Description<Robot>{
public:
  DefDescr(Robot& r):Description<Robot>(r){}
  void getMapMembers(GenericIDPath& hopname,std::vector<NodeMapping*> &members){
    members.push_back(scalarMapping(d(obj.set),IDPath("robot.settings",".")));
    members.push_back(vectMapping(vd(obj.motors),IDPath("robot.motors",".")));
  }
};
```



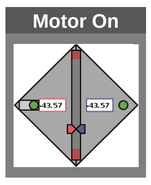

An XML as such on the right can be inputted into the tree by implementing the class as shown above. *scalarMapping* is used to read a single variable (settings) while *vectMapping* can automatically read an array of data (motors). The resulting tree is show on the left.

## Guide

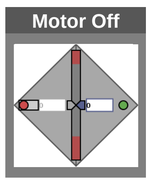



## Joint Status

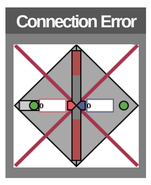
### Motor On



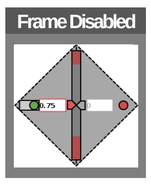

### Motor Off



### Connection Error

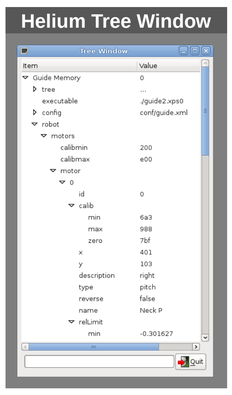


### Frame Disabled

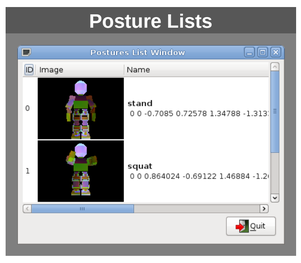


## Other Features

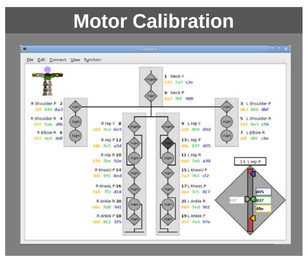
### Helium Tree Window



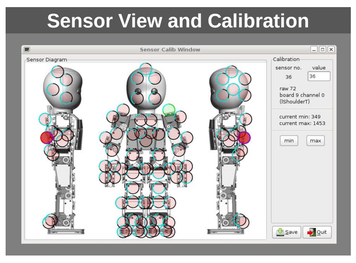

### Posture Lists




### Motor Calibration



### Sensor View and Calibration



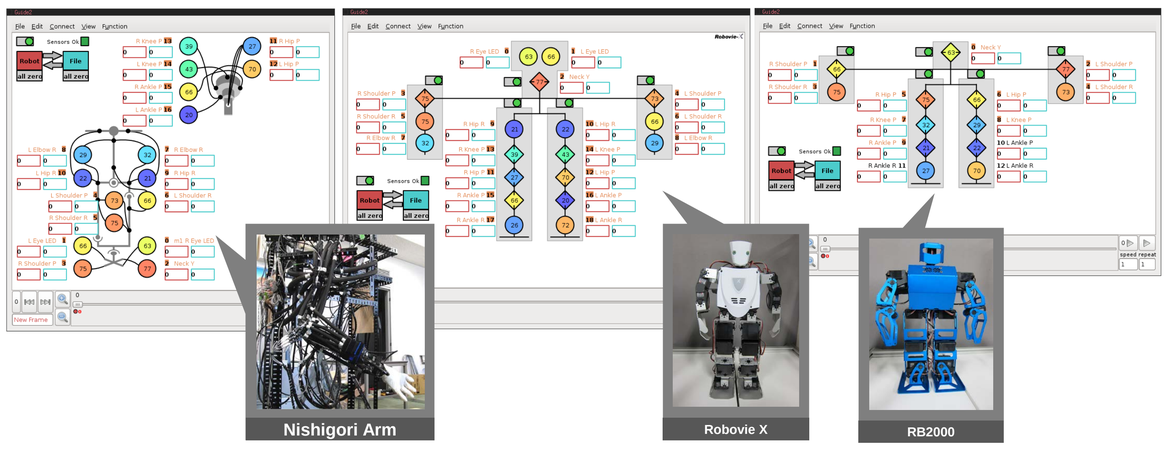

## Dependencies

clapack, gtkmm-2.4, gtkgl-2.0, gthread-2.0, glut

**Tested on**:
Ubuntu 9.10, Ubuntu 11.10
Microsoft Visual C++ 2010

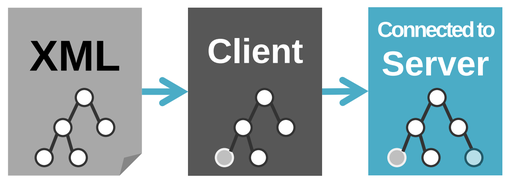
## With other Robots

Just by changing the XML configuration file, the software can be used with other robots without recompiling the program




```
<robot>
 <settings bg="file.png" />
 <motors num="3">
  <motor id="0" x="100" y="200">
   <calib min="6a3" max="988" zero="7bf" />
  </motor>
  <motor id="1" x="200" y="300">
   <calib min="433" max="c2e" zero="7a7" />
  </motor>
  <motor id="2" x="500" y="200">
   <calib min="53d" max="a56" zero="7a2" />
  </motor>
 </motors>
</robot>
```

Client side Tree





[1] Cumming, M., Rieder, B., Jongsma, J., M'Sadoques, J., Laursen, O., Ruebsamen, G., Gustin, C., et al. (2010). Programming with gtkmm 2. Retrieved from http://developer.gnome.org/gtkmm-tutorial/2.24/
[2] Demura, K. (2007). Robot Simulation – Robot programming with Open Dynamics Engine (p. 260). Tokyo: Morikita Publishing Co. Ltd.
[3] Dallalibera, F., Minato, T., Ishiguro, H., Pagello, E., & Menegatti, E. (2009). A software toolset for quick humanoid motion prototyping. Proceedings of the 4th Workshop on Humanoid Soccer Robots, workshop of the 2009 IEEE-RAS Intl. Conf. on Humanoid Robots (Humanoids 2009) (pp. 45-51). Paris, France.
[4] Basoeki, F., Dallalibera, F., & Ishiguro, H. (2010). Analysis of Tactile Instructions Used in the Interaction with a Humanoid Robot. Information Processing Society of Japan Kansai Branch (p. C-02). Osaka, Japan. Retrieved from http://www.ipsj.or.jp/sibu/kansai/ipsj-kansai/hyoshou/document/h22/H22_C-02.pdf