Università degli Studi di Brescia
Facoltà di Ingegneria
Dipartimento di Elettronica per l'Automazione

Behaviours

Ing. Fabio Tampalini

Fonti

- Sono presenti numerosi documenti nella pagina del corso di robotica.

Behavioural Control

- Behavioural actions are implemented in Aria.
- The smaller unit of behaviour is called an ***action***.
- Behavioural actions can the same types of control values as direct actions: traslation velocity and position, and rotational velocity and heading. But, since the outputs are meant to be combined, it is important that the current set of executing actions use the ***same*** control on each of the channels.
- Konolige recommends using translational velocity and rotational heading as the standard action outputs for most uses.

The Action Evaluation Cycle

- The set of currently active actions is held on a list in the robot object **SfROBOT.**
- On every cycle (100ms), each action object is evaluated to produce a translational and rotational output, along with a strength for each.
- The strength, which varie from 0 to 1, indicates how strongly the action prefers to have this motion executed.
- The output values for behavioural actions are described by a structure, **ArActionDesired**.
- There are many possible types of resolution strategies: averaging, winner-take-all, competition, etc.

The Behavioural Action SfMovitAction

- Now we'll examine a behavioural action in detail.
- The porpose of this action is to turn the robot to a given heading, and move it forward a given distance.

```
class SfMovitAction : public ArAction
{
 public:
  SFEXPORT SfMovitAction(int distance, int heading); // constructor
  virtual ~SfMovitAction() {};       // nothing doing
  // this defines the action
  virtual ArActionDesired *fire(ArActionDesired currentDesired);
  // if we need to reset distance to go
  SFEXPORT void reset() { gone = 0; ax = SfROBOT->getX();
                          ay = SfROBOT->getY(); }
  // interface to Colbert
  static SfMovitAction *invoke(int distance, int heading);

protected:
  ArActionDesired myDesired;  // what the action wants to do
  int myDistance, myHeading;  // parameters to the action
  int gone;                   // how far we've gone
  double ax, ay;       // old robot position
};
```

The Action Constructor

```
SFEXPORT
sfMovitAction::sfMovitAction(char *name, bool instance)
    : ArAction("Movit")
{
  myDistance = distance;
  myHeading = heading;
  reset();
}
```

- Note the constructor chaining performed by calling ***ArAction("Movit")*** in the prolog of the constructor.

The Action Body (1)

- The action body is defined by the ***fire()*** function.
- This function is called on every cycle that the action is active, and is responsable for determing the output of the action.
- Remember that this action is supposed to move the robot forward by its first argument, and turn it to the heading given by its second argument.

```
SFEXPORT ArActionDesired *
SfMovitAction::fire(ArActionDesired d)
{
  // reset the actionDesired (must be done)
  myDesired.reset();

  // check the distance to be traveled
  double dx = ax - SfROBOT->getX();
  double dy = ay - SfROBOT->getY();
  ax = SfROBOT->getX();              // set new values
  ay = SfROBOT->getY();
  int ds = (int)sqrt(dx*dx + dy*dy);
  gone += (int)ds;
  sfMessage("Running Movit, gone %d", gone);

  if (gone >= myDistance)
    {
      sfMessage("Finished Movit");
      deactivate();           // turn off when done
      return NULL;
    }
  else
    {
      myDesired.setHeading(myHeading); // control the heading
      myDesired.setVel(200);  // moderate speed
    }
  return &myDesired;          // return the desired controls
}
```

The Action Body (2)

- Once the desired distance has been traveled, the action deactivates itself.
- Deactivation has two effects: first, it frees up the action executive from having to process the ***fire()*** function any more; second, it communicates to Saphira and Colbert that the action has completed.
- If there are other activities that are waiting for the action to complete, they can then proceed.
- If an action does not want to control the movement of the robot, it can return a NULL pointer insted of the ***ArActionDesired*** pointer; this is done when the action finishes.

Interfacing the Action to Saphira/Colbert (1)

- There are two parts to creating the Saphira and Colbert interface to an action. First, the ***invoke()*** function must be define statically, and serves as a function that can be called from Colbert, just as other functions made available with ***sfAddEvalFn***.
- ***Invoke()*** takes arguments that are Colbert types (int, floats, and void * types), and return an istance of the action.

```
SfMovitAction *
SfMovitAction::invoke(int distance, int heading)
{
  return new SfMovitAction(distance, heading);
}
```

Interfacing the Action to Saphira/Colbert (2)

- In the file defining the action, we can add code to be executed when the file is loaded into Saphira, using then function ***sfLoadInit***.
- Here is how it's done for ***AfMovitAction***:

```
SFEXPORT void                    // define interface to Colbert here
sfLoadInit ()
{
  sfAddEvalAction("Movit", (void *)SfMovitAction::invoke, 2, sfINT, sfINT);
}
```

Invoking the Action (1)

- Once the action is defined and added to Saphira and Colbert, it can be invoked from Colbert using the ***start*** command, just like a Colbert activity.
- To start up the action, just use

```
start Movit(1000,95);
```

which will start up the action with a distance argument of 1000 and a heading argument of 95.

- An example of invoking the ***Movit*** action from Colbert is given in the ***tutor/movit/init.act*** file, as the activity ***movit_act***.

```
act movit_act(int dist)
{
  start Movit(dist, 90) priority 10;
  remove Movit;                  // get rid of the action
  stop;                          // stop the robot
}
```

Invoking the Action (2)

- From C++ programs, the action can be invoked by using the corresponding ***sfStartTask()*** function. For example, here a tipycal invocation:

```
sfStartTask("Movit", NULL, 0, 10, 0, 1000, 95);
```

- Here the second argument is the instance name; the default NULL means it will be the same as the schema name. Timeout, priority, and suspension arguments are required, followed by the action parameters.

Turning Behavioural Action On

- Behavioural actions and direct actions can conflict if they are invoked at the same time.
- In situation like this, the direct actions always take priority.
- While the robot is executing direct actions, it turns off the output of behavioural actions.
- In fact, whenever a direct action is executed, behavioural actions remain off untill they are explicitly turned back on.
- To turn bahavioural actions on, use the ***behaviors*** command in Colbert, or the longer C++ function:

sfROBOT->clearDirectMotion()

FINE