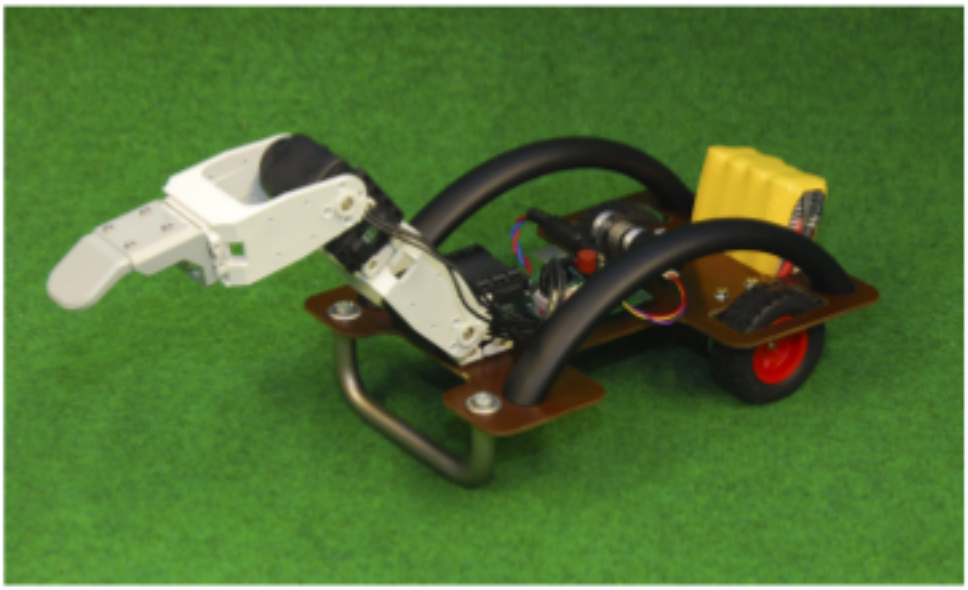

Ravensburg-Weingarten University of Applied Sciences
iki.hs-weingarten.de

Crawler Manual

November 28, 2012

Contents

1 Features

We present a little crawling robot with a two DOF arm that learns to move forward within about 15 seconds in real time. Due to its small size and weight the robot is ideally suited for classroom demonstrations as well as for talks to the public. Students who want to practice their knowledge about reinforcement learning and value iteration can use a wireless connection to a PC and monitor the internal state of the robot such as the value function or the reward table. Due to its adaptivity, depending on the surface properties of the underground the robot may surprise its audience with unexpected but efficient walking policies. The GUI is available as an OpenSource project.

Our crawling-robot prototype is controlled by an ATmega32 microcontroller board which is mounted on top of the robot. The joints of the robot are driven by servos and the robots movement is measured by an optical incremental encoder attached to one of the wheels. The board has outlets for the servos, a serial wireless transceiver, an outlet for the encoder and a DIP switch to setup several parameters. For instance one of these parameters inverts the encoder signal such that the robot learns a backward-moving strategy instead of moving forward. For more inforamtion about the different states see Chapter 2.

1.1 Theoretical Introduction

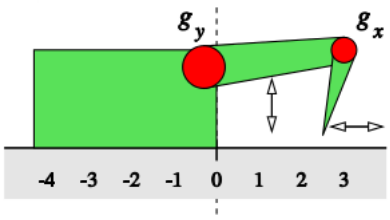

Figure 1: The Robot with its two joints g_x and g_y

We consider the reinforcement learning framework, where an agent, in our case the crawling robot, interacts with a Markovian decision process (MDP). At each time $t \in \{0, 1, 2, \dots\}$ the agent is in a certain state $s_t \in S$. After executing action $a_t \in A$ the agent receives a reward signal $R^a_{ss'} = E\{r_{t+1}|s_t = s, a_t = a, s_{t+1} = s'\}$ by passing into a successor state s_{t+1} with a probability $P^a_{ss'} = Pr\{s_{t+1} = s'|s_t = s, a_t = a\}, \forall s, s' \in S, a \in A$. $\delta(s, a)$ denotes a transition function and represents the successor state s' after action a has been selected in s. The decision which action a is chosen in a certain state s is characterized by a policy, $\pi(s) = a$, which could also be stochastic $\pi(s, a) = Pr\{a_t = a|s_t = s\}$. A policy which maximizes the average reward over time is denoted as $\pi*$. On the crawling robot we use the value-iteration algorithm for learning a cyclic deterministic policy $\pi(s)$ by which the robot moves forward. The algorithm value iteration which we use on the robot basically works by assigning a numerical value $V(s) \in V$ to each state $s \in S$ where each state value represents the expected cumulated reward over time when following $\pi(s)$. Based on these state values V in conjunction with $R^a_{ss'}$ deterministic policy $\pi(s)$ can be derived.

We adapted the value-iteration algorithm to perform on the crawling robot as depicted in Algorithm 1. Here a discounting factor $0 < 1$ is used to specify the portion of influence of neighbor-state values $V(\delta(s, a))$ on $V(s)$. Since the robot is faced with a zero-knowledge environment after switching it on, a tradeoff between exploration (longterm optimization) and exploitation (short-term optimization) has to be done. A very simple exploration technique is ϵ-Greedy exploration. Here the agent selects an action at random with a probability ϵ, uniformly and independently of $V(s)$, instead of following $\pi(s)$. Heuristically ϵ could be set to a high value at the beginning of the learning process and then be decreased over time. This ensures much exploration at the beginning and pure exploitation starting at some time t. As the value-iteration algorithmr equires the rewards $r(s, a)$ for each action a in state s, we simply save them into a memory table $R^a_{ss'}$. After executing action a in s, we update the corresponding record $r(s, a)$ in $R^a_{ss'}$.

Algorithm 1 VALUEITERATION ON ROBOT

Initialize V arbitrarily, e.g., $V(s) = 0$, for all $s \in \mathcal{S}$
Initialize $\mathcal{R}^a_{ss'}$ arbitrarily, e.g., $r(s,a) = 0$, for all $r \in \mathcal{R}^a_{ss'}$
$state \leftarrow (g_x = 1, g_y = 1)$
loop
    $\xi \leftarrow \text{rand}(0..1)$
    if $\xi < \varepsilon$ **then**
        $a \leftarrow \text{rand}(\mathcal{A}(state))$
    else
        $a \leftarrow \text{argmax}_a \mathcal{R}^a_{ss'} + \gamma V(s')$
    end if
    $successorState \leftarrow \delta(state, a)$
    observe $r(state, a)$ and update $\mathcal{R}^a_{ss'}$
    for all $s \in \mathcal{S}$ **do**
        $V(s) \leftarrow \max_{a \in \mathcal{A}(s)} \Big[r(s,a) + \gamma V\big(\delta(s,a)\big) \Big]$
        $\pi(s) \leftarrow \text{argmax}_{a \in \mathcal{A}(s)} \Big[r(s,a) + \gamma V\big(\delta(s,a)\big) \Big]$
    end for
    $state \leftarrow successorState$
end loop

2 The DIP switch

You can find 8 switches on your Crawler board. Each Switch has 2 different stages, ON and OFF. The following table explains the function of them.
Hint: It my take some time that the changes take effect.

Number	ON	OFF
1	The Crawler explores and learns.	The Crawler is in PAUSE-Modus. It stops learning but doesn't forget the knowen things
2	The Crawler performes multiple actions to go in an other state and than checks the Result.	The Crawler performes only a single action an than checks the Result.
3	Learns to walk forward	Learns to walk backward
4	Exploration/Exploitation. The Crawler explores the environment with random actions. See Switch 7 to change the Exploration behavior	(pure) Greedy Exploration. The Crawler takes the best action in his movement table.
5	The Crawler explores his environment and stops when he's done.	Just starts exploration/exploitation on start.
6	Turns the serial interface on. (See Chapter 4. The serial interface)	Turns the serial interface off. (See Chapter 4. The serial interface)
7	The robot uses Value-Difference Based Exploration. This is method for balancing the exploration/exploitation dilemma. It adapts the exploration parameter of -greedy. Take a look at 1.1 on page 2	The robot uses a constant -greedy value (= 0.075)
8	Set gamma = 0.98	Set gamma = 0.90

3 The LEDs

The LED Color depends on the Crawlertype (Big Crawler or Small Crawler)

Number	Color(Big/Small)	ON	OFF
1	Red/Red	The crawler is on	The crawler is off
2	Yellow/Green	The Crawler is performing an actison.	The Crawler is not performing any action.
3	Red/Green	The Crawler is learning currently.	The Crawler is not learning currently.
4	Green/Green	The Crawler explores his environment.	The Crawler doesn't explore.

4 The Grafical User Interface [BETA]

The Grafical User Interface is part of the Teachingbox which is OpenSource and available at http://sourceforge.net/projects/teachingbox/

The Crawlerpart is located at /bin/org/hswgt/teachingbox/crawler

There are predefined Usecases. To start a Usecase you have to copy the TeachingBoxAPI in your java-path. Normaly java is located at /usr/lib/jvm/<java-Version>/jre/lib/ext. After that create a start-script (e.g. start.sh) with the following Content:

```
#!/bin/bash
dependencies
# This is a shorthand for the call of a main-function within a class-file
# Example ./start.sh org.hswgt.teachingbox.crawler.usecases.GridworldSimuExec


# The Java interpreter
JAVA=/usr/lib/jvm/java-1.5.0-sun/bin/java

CLASSPATH=bin:\
contrib/UMLGraph/lib/UmlGraph.jar:\
contrib/log4j/lib/log4j-1.2.15.jar:\
contrib/colt/doc/api/hep/aida/ref/doc-files/aidaref.jar:\
contrib/colt/lib/colt.jar:\
contrib/colt/lib/concurrent.jar:\
contrib/colt/src/hep/aida/ref/doc-files/aidaref.jar:\
contrib/junit/lib/junit-4.5.jar:\
contrib/jogl/lib/gluegen-rt.jar:\
contrib/jogl/lib/jogl.jar

${JAVA} -cp ${CLASSPATH} $1
```

You can now start a Usecase with ./start.sh org.hswgt.teachingbox.crawler.usecases.<Usecase> (e.g: ./start.sh org.hswgt.teachingbox.crawler.usecases.GridworldSimuExec

5 The Serial Interface

The Serial Interface is necessary to let the Crawler communicate with the GUI.

Hint: We do not ship the Crawler with the Serial Interface yet

6 Programming the Crawler

The Amiga32 Controler is fully programmable. The Sourcecode for the Crawler is OpenSource and available at ailab.hs-weingarten.de/crawler-code.tgz. Feel free to modify it!

6.1 Compile the Microcontroller Code

If you want to recompile the Microcontroler Code you need EXECTLY the folloing packages:

- binutils-avr_2.15-3_i386.deb (Download: http://archive.debian.net/uk/sarge/i386/binutils-avr/download)
- avr-libc_1.2.3-3_all.deb (Download: http://archive.debian.net/de/sarge/all/avr-libc/download)
- gcc-avr_3.4.3-2_i386.deb (Download: http://archive.debian.net/uk/sarge/i386/gcc-avr/download)

The Packages doesn't have any dependencies so you can install them on any Debian System.

6.2 Changing the BioloidServoID (Big Crawler only)

We use 2 BioloidServos for the arm of the big crawler. The default IDs are: 2 for the crawler directly on the board and 6 for the one at the end of the arm. To change them you have to modify the sourcecode. Find the "Environment.c"-File and go to line 30/31. This is the place where the Code creates the handles for the Servos. servo_x is the upper servo, servo_y is the one which is directly on the board. Change line 30/31 to modify the IDs and than recompile the microcontroller code (see 6.1).

7 Battery

You can use every Batterie with an output from 7 to 10 Volt. We ship our Crawler with a 9 volt battery or with a 7.4 volt LiPo battery