Markov Crawler
Users Manual

jan.balewski@gmail.com, ver 1.0

December 25, 2017

This instruction assumes you have assembled the Markov Crawler Kit following the Assembly Instruction. You can ssh to Raspberry Pi (R-Pi) from your laptop (or a desktop) connected to the same local network as R-Pi WiFi. The muSD card included in Markov Crawler Kit contains the preinstalled Rasbian OS and preinstalled Markov Crawler software.

Contents

Listings

1 Ssh to Raspberry Pi (R-Pi)

Power ON R-Pi , the red LED on R-Pi should stay on, the green one (disc access) should blink few times, then turn off. In this section I'm assuming your laptop runs Linux-like operating system. It is assumed you have basic Linux skills, you know Python, and you have some experience with R-Pi . Open an x-terminal your laptop. Execute those commands:

1. to check if R-Pi is visible from the laptop 'ping' it using the local IP you have identified when R-Pi was connected to the physical terminal during the assembly process. Execute:

```
laptop1$  ping -c 2 192.168.1.6
PING 192.168.1.6 (192.168.1.6): 56 data bytes
64 bytes from 192.168.1.6: icmp_seq=0 ttl=64 time=560.956 ms
64 bytes from 192.168.1.6: icmp_seq=1 ttl=64 time=10.104 ms
```

2. use ssh to connect to R-Pi, include opening the graphic tunnel by adding '-X'. The R-Pi user name is **pi**, password **jas**, user pi is sudo. Execute:

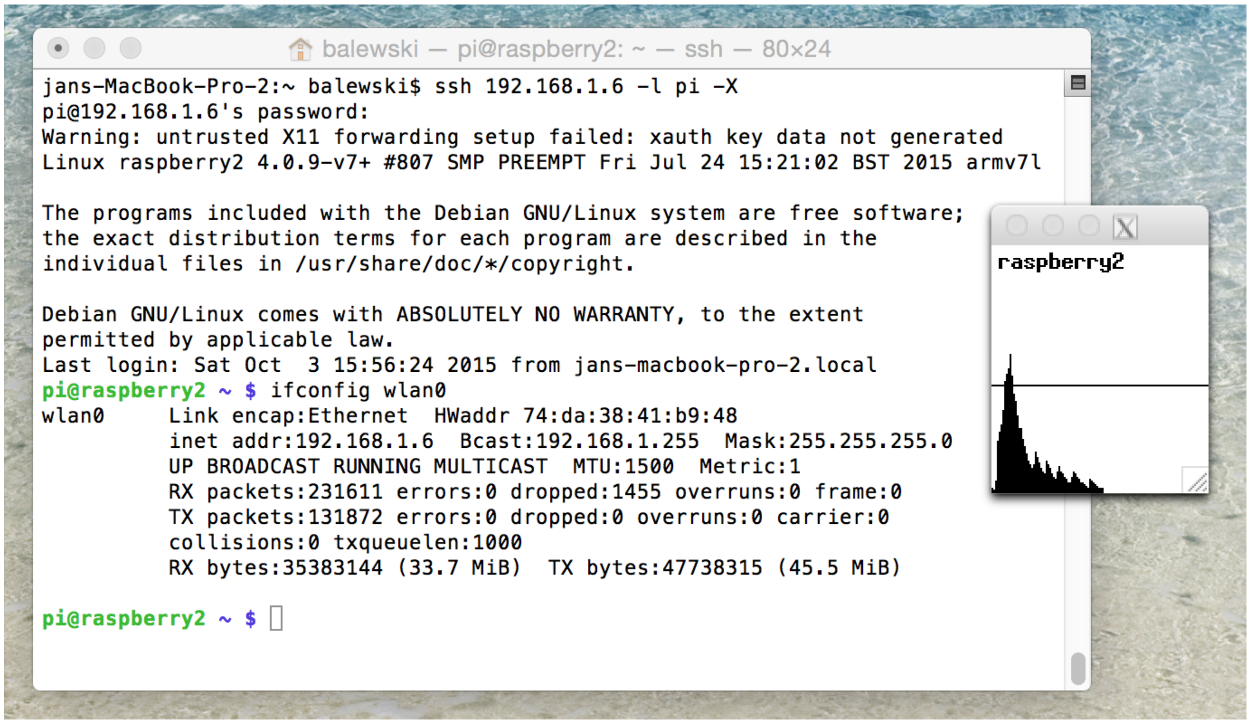
```
laptop1$  ssh 192.168.1.8 -l pi -X
pi@192.168.1.6's password:
....
Linux raspberry2 4.0.9-v7+ #807 SMP PREEMPT Fri Jul 24 15:21:02 BST 2015 armv7l
```

```
.......
$ cd
$ ls -F
 markov-crawler/  Desktop/  Adafruit-Raspberry-Pi-Python-Code/
```

3. verify graphics works by executing on R-Pi:

   ```
   $ xload &
   ```

   The display of load vs. time should pop up on your laptop in few seconds, see Fig. ??.



```
jans-MacBook-Pro-2:~ balewski$ ssh 192.168.1.6 -l pi -X
pi@192.168.1.6's password:
Warning: untrusted X11 forwarding setup failed: xauth key data not generated
Linux raspberry2 4.0.9-v7+ #807 SMP PREEMPT Fri Jul 24 15:21:02 BST 2015 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Oct  3 15:56:24 2015 from jans-macbook-pro-2.local
pi@raspberry2 ~ $ ifconfig wlan0
wlan0     Link encap:Ethernet  HWaddr 74:da:38:41:b9:48
          inet addr:192.168.1.6  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:231611 errors:0 dropped:1455 overruns:0 frame:0
          TX packets:131872 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:35383144 (33.7 MiB)  TX bytes:47738315 (45.5 MiB)

pi@raspberry2 ~ $
```

Figure 1: On your laptop open x-terminal, ssh to R-Pi . To verify the graphics is transported from R-Pi to laptop launch the 'xload' application monitoring the load on the R-Pi .

1.1 Update Markov Crawler software

The Markov Crawler software is stored in the R-Pi directory **/home/pi/markov-crawler/**. If this directory does not exist (or was erased by accident) you can always pull a fresh copy from the Bitbucket Git as described in the Appendix **??**.

It is likely there will be an update to the Markov Crawler software since the Markov Crawler Kit was sent to you, so before you start modifying your Markov Crawler code please resynchronize it with Git by executing:

```
pi@raspberry2 ~ $ cd markov-crawler
pi@raspberry2 ~/markov-crawler $ git pull
```

```
......
Unpacking objects: 100% (3/3), done.
From https://bitbucket.org/balewski/markov-crawler
   0271b9e..b074f37  master     -> origin/master
Updating 0271b9e..b074f37
....
 1 file changed, 1 insertion(+), 1 deletion(-)
```

1.2 Markov Crawler code structure

The Markov Crawler code is written in Python (runs under Python ver 2.7). A significant part of the code was borrowed from the Berkeley CS-188 EdX online class. The CS-188 code was refactored to integrate it with the hardware connected to R-Pi , several new modules were added. You should see the following subdirectories:

```
$ cd
$ ls
Desktop/       Adafruit-Raspberry-Pi-Python-Code/       markov-crawler/
```

The **Adafruit-Raspberry-Pi-Python-Code/** directory contains a low level Python code needed to control the PWM pulse generator. There is no need to modify it. You will work with the code localized in the **markov-crawler/** directory. Let see what is there?

```
 $ cd ~/markov-crawler
 $ ls -F
 mouseUtil/          reflexAgent-cs188/  setup@       utils-cs188/
qlearnAgent-cs188/  reflexAgentSimple/  setup-RPi1/
Readme.txt          servosUtil/
```

- **setup/** directory contains various constants pertaining to the Markov Crawler hardware, e.g. pickle defines the angular range for servo-motors, calibrates mini-mouse, defines dimensions of physical Markov Crawler used by in the virtual model.
- **servosUtil/** - here resides the servo driver (servoDriver.py) and the servo calibration tool (servoCalibMain.py) which you will use next.
- **mouseUtil/** - here resides **mouseDriver.py** code and a calibration tool **mouseCalibMain.py**.
- **reflexAgentSimple/** - can make the Markov Crawler to walk using predefined sequence of rotations of servos, see more in Section. **??**.
- **reflexAgent-cs188/** - state-machine based interface to Markov Crawler . A GUI allows for execution of individual steps as well as of a pre-defined policy. The animation of Crawler pose and readout of its measured position is displayed.
- **qlearnAgent-cs188/** - the AI brain of the Markov Crawler . Uses Markov decision process to train Crawler to walk either virtually or based on optical mouse feedback.

2 Calibration of servo motors

A servo motor can usually only turn for a total of 180^o. Servo motors are controlled by an electrical pulse of variable width, sent by the PWM generator via one of the 3 servo wires. The PWM pulse width is expected to be between 150 and 650 (μs), the repetition rate is fixed, set to 50 Hz. [1].

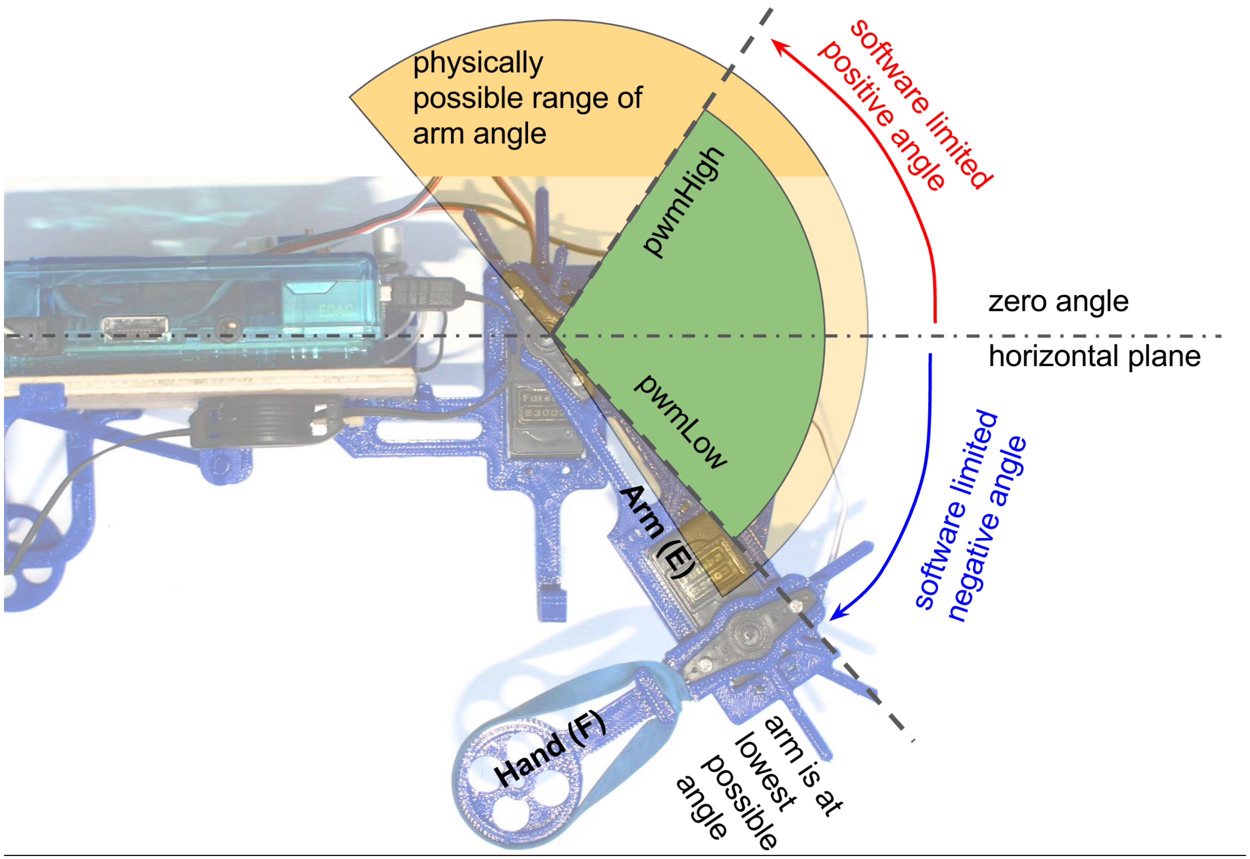

physically possible range of arm angle
software limited positive angle
pwmHigh
zero angle
horizontal plane
pwmLow
software limited negative angle
Arm (E)
Hand (F)
arm is at lowest possible angle

Figure 2: Physical and software angular limits of the arm of the Markov Crawler . Similar limits apply to the hand angle.

? During the assembly you have rotated both servos clock-wise (w/o power) to such position that the arm and the hand are at the extreme but do not touch anything. This is 'down' position of both servos. Now we need to define in software the lower and upper angu?lar limits.

The end goal is to translate the desire angle of an arm/hand expressed in radians in to the physical position. This is the generic formula converting the requested angle α measure in radians to the PWM value resulting with desire rotation of the servo.

$$x(\alpha) = pwm_0 + dir \cdot \alpha \cdot fact, \quad \text{where} \quad -\pi/2 < \alpha < +\pi/2 \tag{1}$$

$$pwm(\alpha) = \begin{cases} pwm_H & \text{if } x(\alpha) > pwm_H \\ pwm_L & \text{if } x(\alpha) < pwm_L \\ x(\alpha) & \text{otherwise} \end{cases} \tag{2}$$

[1] http://www.jameco.com/jameco/workshop/howitworks/how-servo-motors-work.html

The servo driving software needs to know the 'configuration': $dir = \pm 1, fact, pwm_0, pwm_L, pwm_H$ which depend on the particular servo you are using and on the angle you inserted the servo horn on the servo shaft. Fig. **??** will help you to understand the relations between those parameters.

You will find all those constants experimentally with the help of the GUI shown in Fig. **??**. The servo configuration will be saved in to a Python pickle file, separately for the arm and for the hand.

Launch the **servoCalibMain.py** program calibrating servo as follows. Note, you need launched this code as 'gksu' [2]) to have enough privileges to access R-Pi GPIO pins communicating with the PWM generator via IC2 protocol.

```
$ cd ~/markov-crawler/servosUtil
$ gksu "./servoCalibMain.py"
actConfig action= read
reading pickle from : ../setup/generic.servo.conf
....
```

The GUI window (see Fig. **??**) should pop-up on your laptop. By default, this program reads in the 'generic' servo params **../setup/generic.servo.conf** which are incorrect. If needed, you can change this input pickle name at the launch time by adding run-time parameters (use -h flag to see all options). Before we proceed read those few comments:

- the toggle switch (Fig. **??** bottom-left) can be used at any time to turn ON/OFF the power to servos - if they seems to behave out of control. The power to R-Pi is not interrupted in such case and you should not loose connection to it.

- the A-meter shows the sum of current drawn by both servos. It is expected to see below 0.90A per servo, only if they move and encounter a resistance (or up to 1.8A if both servos move simultaneously). If you observe large current drawn when servo is not moving try to figure out what is blocking the servo. If you can't find it within a minute, then toggle the servo power switch to OFF and continue thinking. You do not want to dissipate that much power in to a servo for more than few minutes. To give you the scale 5V*1.8A=9W of power.

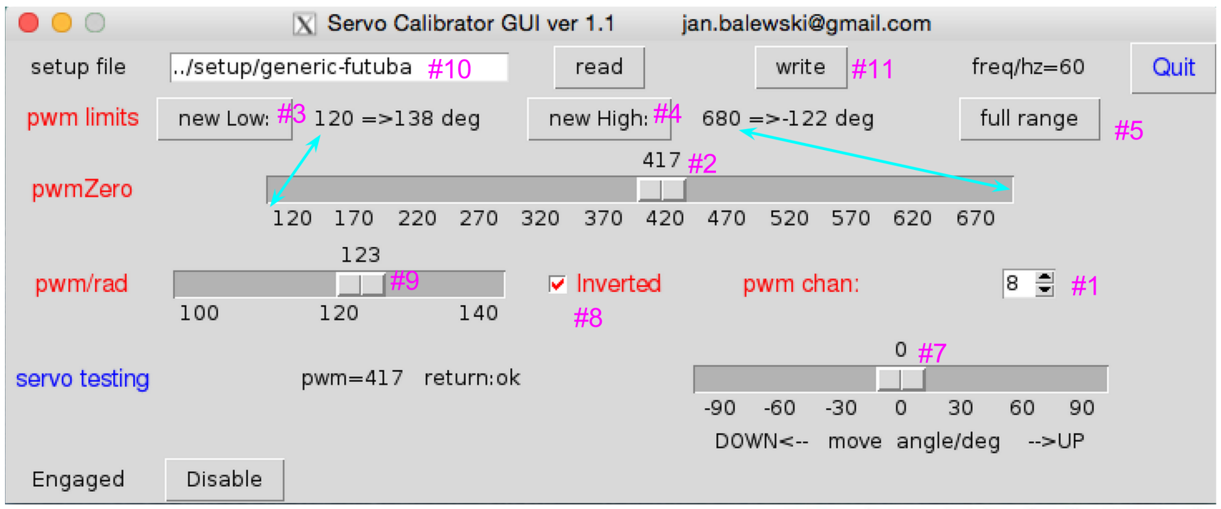
- there are magenta #-numbers added to Fig. **??** to guide you through the calibration process. In general, all red-labeled values are stored in to the pickle, meaning do not touch those sliders until you press 'write'. The black-labeled values are only to help you to calibrate the servo, meaning it does not matter what those values are when you press 'write'.

- the order you change sliders matters for the calibration constants, so move only the slider you are asked to move.

- at any time the PWM value computed using Eq. **??** and sent to the servo is displayed in the GUI at location #12.

For each servo you will need to repeat the following steps. Lets start with the 'arm' servo, manually bent the 'hand' vertically up.

[2]Gksu is a tool to allow a user to run a GUI program as root. It does for X programs what sudo does for commands.


Servo Calibrator GUI ver 1.1 jan.balewski@gmail.com

setup file ../setup/generic-futuba #10 read write #11 freq/hz=60 Quit

pwm limits new Low: #3 120 =>138 deg new High: #4 680 =>-122 deg full range #5

pwmZero 417 #2
120 170 220 270 320 370 420 470 520 570 620 670

pwm/rad 123 #9
100 120 140

[✓] Inverted #8 pwm chan: 8 #1

servo testing pwm=417 return:ok

0 #7
-90 -60 -30 0 30 60 90
DOWN<-- move angle/deg -->UP

Engaged Disable

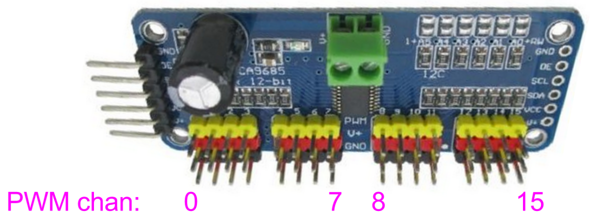


Figure 3: GUI allowing to calibrate each servo motor. The numerical values for each servo will be different.

1. confirm the 3-prong arm-servo cable is plugged into PWM channel 7 (see Fig. ?? bottom-right). The black cable wire should be plugged to black ground pin on the PWM controller.
2. select 'pwm chan' value to be 7 (GUI location #1), the arm should jerk
3. slowly move 'pwmZero' slider (location #2) - the hand should rotate over whole physically allowed 180^0 range.
4. move slider #2 to set the arm angle of at about 45^0 UP, above the horizontal plane.
5. press 'newLow' button (location #3) - the value of slider #2 should show up behind this button. If you move slider #2 to lower value the hand will not move more. You have set the soft limit for the maximal UP-angle of the arm.
6. move slider #2 in the opposite direction to set arm angle at about -45^0 (DOWN), below the horizontal plane.
7. press 'new High' button (location #4) - the value of slider #2 should show up behind this button.
8. if you want to reset the angular range set by buttons #3 & #4 press the 'full range' button (location #5). Next, repeat setting of both ranges. It may happen that for your servo the UP/DOWN and High/Low relation is reversed. It is fine - set the angular limits so they make sense.
9. move slider #2 to set the arm horizontally and do not move this slider again.

10. until now the 'setAngle/deg' slider (location #7) was at the value of 0^0. If not. fix it and start all over.

11. move slider 'angle/deg' (location # 7) to $+30^0$ (UP) and verify the arm moved UP. If it moved down then (un) check 'Inverted' (location #8). This should fix the problem.

12. now we want to fine-tune the relation between set angle and the servo angle. If the arm is significantly off $+30^0$ use slider 'pwm/rad' (location #9) to improve it. To verify the $fact$ variable in Eq. **??** is correct move the slider #9 to -45^0 and check if the arm angle is now -45^0.

13. you are almost done. Edit the pickle name, change 'generic' to 'arm' in the 'setup' file field (location #10) and press 'write' button (location # 11). You should see on the x-terminal:

    ```
    actConfig action= write
    write pickle : ../setup/arm.servo.conf
    ```

14. quit GUI by pressing 'Quit' button (location #13). This GUI runs on a separate thread so crtl-C will not kill it.

The steps above generated **arm.servo.conf** configuration pickle for the arm. Next, repeat the same calibration procedure for the hand servo with the following changes:

- set the hand angle range to be $[-150^0, +30^0]$ instead of the arm $[-45^0, +45^0]$.
- the hand servo is connected to channel 8 of PWM and you need to change it in GUI at the location #1.
- save the pickle as **hand.servo.conf**

You may want to verify the calibration you are using. To do so you can upload the existing pickle to GUI by entering the path-to and name of the pickle at location #10 and pressing the 'read' button (location #14). Alternatively, you can specifying the path-to and name of the pickle at a GUI launch time:

```
$ gksu "./servoCalibMain.py --setupPath ../setup/ --name arm"
actConfig action= read
reading pickle from : ../setup/arm.servo.conf
  ...
```

To conclude, verify both pickle files exist and are of non-zero length:

```
 ls~/markov-crawler/setup/generic.servo.conf   arm.servo.conf   hand.servo.conf
```

2.1 Exercise one servo with moveOneServo.py

Now it is time to exercise the servo calibration you have just established. Inspect the short Python test code **moveOneServo.py**, shown in a compact form in Listing **??**.

```
 1 $ cat moveOneServo.py
 2 import time, os
 3 import pickle
 4 from servoDriver import ServoDriver
 5 
 6 confName =”../setup/hand.servo.conf”
 7 #confName =”../setup/arm.servo.conf”
 8 
 9 conf=pickle.load( open(confName, ”r” ) )
10 
11 servo = ServoDriver()
12 servo.setupController()
13 servo.config(conf)
14 
15 for ang in [ 0., 30., −30, 0.]:
16     servo.setAngleDeg(ang)
17     print 'set angle(deg)=%s, status=%s '%(ang,servo.status)
18     time.sleep(0.8) # important to give servo some time to reach new position
19 
20 servo.fullStop()  # now A−meter will show no current
```

Listing 1: moveOneServo.py python code rotating hand-servo over list of angles.

Lets walk through key lines of the Listing **??** allowing to position one servo at the angle given in degrees.

- line 4: imports the servo driver module
- line 6: defines location of the pickle with calibration of hand-servo. Note, by changing the name of pickle to the arm (commented out line 13) you would control the arm-servo instead.
- lines 11-13 : servo module is initialized with calibration from the pickle file
- line 15 defines loop over list of angles of your choice.
- line 16 : servo1.setAngleDeg(ang) will tell servo to change change to new angle, specified in deg. Alternatively, you can use angle in radians and call : servo.setAngle(-3.14/4) .
- line 17 shows how servo is reporting if your request is sensible. If you command an angle beyond the soft range you have chosen during calibration then servo will stop at the boundary and return string 'low' or 'high'.
- line 18 gives servo time to reach new position, program must simply wait
- line 20 shuts down PWM generator and servo does not hold its position. Until Servo-Driver:fullStop() is executed servo can draw (sizable) current.

Execute the code :
$ gksu ./moveOneServo.py

and observe the hand-servo moves. Next, grab Crawler hand with your hand, so servo must work harder and observe the current shown by A-meter is approaching 0.9 A. It is OK for a servo to draw that much current for few seconds, but make sure it is not constant for an hour - this would overheat/destroy the servo. Finally, change the name of the calibration pickle to the arm-servo (enable line 7) and verify now the arm moves.

Fell free to modify the **./moveOneServo.py** code and make it more complex. Note, ServoDriver instance talks to one PWM, so you can't have 2 instances of ServoDriver. The next section shows how to drive 2 servos simultaneously.

3 Programmatic arm-hand coordination (Reflex Agent)

This section will discuss how to control and coordinate movement of the hand and the arm of the Markov Crawler to cause Markov Crawler to walk forward. We will start with the simplest approach, by extending the one servo example to two servos. Next, we will use two correlated harmonic oscillator functions to encode more flexible and more smooth but still predefined motion pattern of the Markov Crawler . Finally, we will adopted the State Machine based code from CS-188 to encode walking which will be encode list-based (again), but the code will be ready to be interfaced with the Q-learning AI code from SC-188, discussed in the next section.

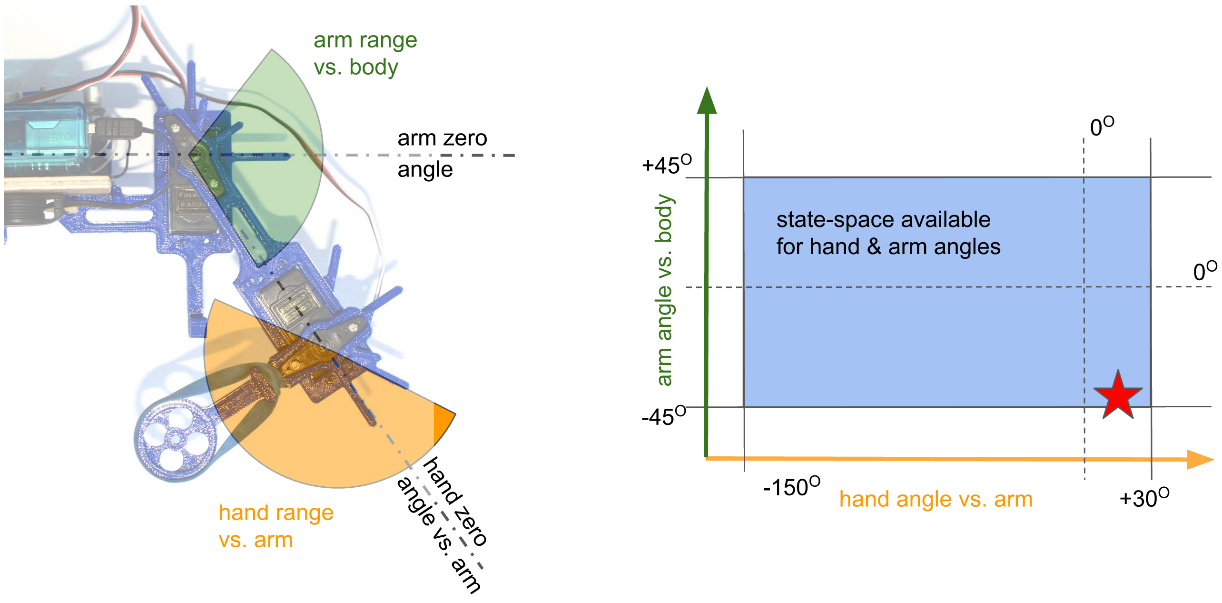


Figure 4: Left: angular range allowed for Crawler hand and arm. Right: the same range shown as rectangle. The red star is equivalent to the arm and hand position shown on the left.

Before discussing the Python code let introduce the notion of the arm-hand state space. Fig. **??** -left shows the angular range allowed (in software) for both appendages. Note, the hand angle is defined relatively to the arm 'bone'. Fig. **??**-right shows all possible hand and arm angles values on a 2D plane. Any physical configuration of the Markov Crawler can be represented as a point inside the blue rectangle. E.g. the red star is equivalent to the arm and hand position shown on the left.

3.1 Discrete list-based walking

The simplest reflex agent uses a list of (arm,hand) angle values, as shown in Fig. **??**-left. It loops over the list in predefined order, dials the pair of angles, and waits for a fraction of second until servos reach the new set position. The movement is jerky because torque of the servos is strong. Listing. **??** shows full implementation of this algorithm. Execute the code :
$ gksu ./moveOneServo.py
and observe the hand-servo moves

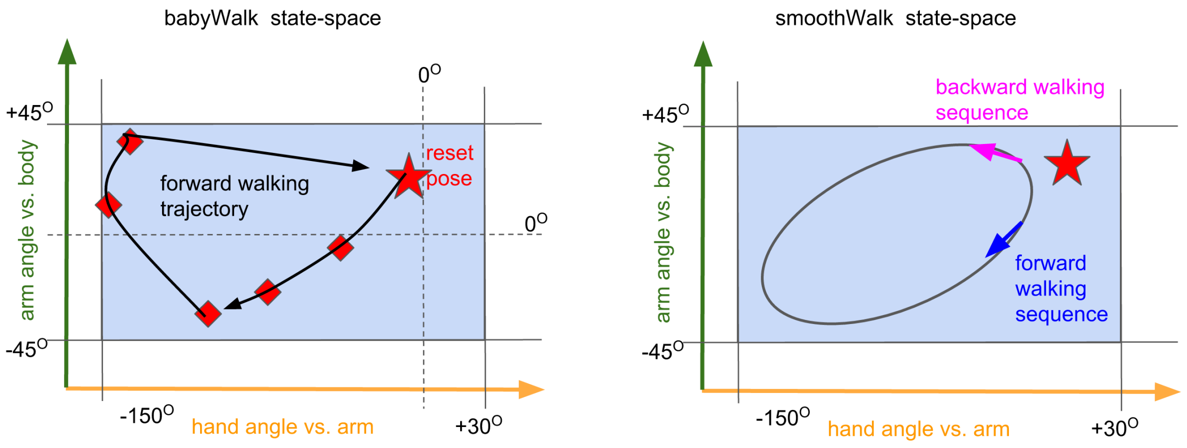


Figure 5: Left: Red points mark trajectory implemented in babyWalk reflex agent. Right: smoothWalk reflex agent uses two correlated harmonic oscillator functions for more smooth walking. By changing the relative phase one can reverse the direction of walking or stale the Crawler.

```
$ cat babyWalk.p
import time
import pickle
import sys, os
sys.path.append(os.path.relpath("../servosUtil/"))
from servoDriver import ServoDriver

# define parameters below
stepDelay=0.3 # (seconds), between steps
walkTime=10. #  (sec) , total duration of walking
handConfName="../setup1/hand.servo.conf"
armConfName ="../setup1/arm.servo.conf"
dbg=0  # debugging switch , set to 1 to see more printouts

# open pickles for hand and arm and initialize two instances of ServoDriver
handConf=pickle.load( open(handConfName, "r" ) )
hand = ServoDriver(dbg)
hand.setupController()
hand.config(handConf)

armConf=pickle.load( open(armConfName, "r" ) )
arm = ServoDriver(dbg)
arm.linkController(hand.pwmDriver) # use common PWM driver
arm.config(armConf)

```

```
26 #define reset state of both servos
27 stateReset=(20,-15) # (arm,hand) angles in deg  at reset position
28 arm.setAngleDeg(stateReset[0]) #executes arm reset
29 time.sleep(0.5) # this delay desynchronizes power consumed by hand from arm,
       keep it
30 hand.setAngleDeg(stateReset[1])
31 time.sleep(0.5)
32 
33 # definition of (arm,hand) relative angular states  in deg for one cycle
34 stateList=[stateReset,(-5,-45),(-20,-80),(-40,-100),(-5,-150),(30,-140)]
35 
36 # execution of cycle until time runs out
37 while walkTime >0 :
38     print "------walking time left %.1f (sec) "%walkTime
39     for state in stateList:
40         arm.setAngleDeg(state[0])
41         hand.setAngleDeg(state[1])
42         time.sleep(stepDelay)
43         walkTime=walkTime - stepDelay
44 
45 hand.fullStop()  # must be executed to free servos
```

Listing 2: **babyWalk.py** python code sequencing through the list of pair of angles for hand and arm causing Crawler to walk along the trajectory shown in Fig. **??**-left.

The new elements in Listing. **??** are discussed below.

- lines 9,10 : define the time between steps and total duration of the walk
- line 23 : show how 2nd servo can be added to the same PWM controller
- lines 26-31 : show how to initialize both servos from unknown position at cold start w/o drawing too much current (which could cause R-Pi to reboot)
- line 34 : defines the sequence of angle pairs for one cycle of walking (shown as red points in Fig. **??**-left)
- lines 40-42 : implement simultaneous movement of both servos

3.2 Continuos pendulum-based walking

A more smooth motion can be implemented by moving both servos more frequently by a smaller angle, as shown in Fig. **??**-right. Mathematically, we want to model the amplitudes of the angles of the hand and the arm by two harmonic oscillator functions which relative phase will decide on the walking direction.

$$\phi_{\rm arm} = \phi_0 + \phi_{\rm ampl} \sin(\omega \cdot \Delta t \cdot k) \qquad (3)$$

$$\theta_{\rm arm} = \theta_0 + \theta_{\rm ampl} \sin(\omega \cdot \Delta t \cdot k + \alpha) \qquad (4)$$

where $\omega \cdot \Delta t \cdot k$ controls how both amplitudes change with time encoded as micro-steps k. The relative phase α controls if Crawler walks forward, backward, or does not move at all. The key element of this algorithm are shown in Listing. **??**. The full code is available at the directory ∼**/markov-crawler/reflexAgentSimple**.

```
arm_hand_phase=80 #deg
walkTime=10. #  (sec) , total duration of walking
stepDelay=0.05 # (seconds), between micro-steps
rotation_omega=250  # deg/second

# define  mean angle and amplitude of oscillations , units: deg
arm_phi0=-10
arm_ampl=30
hand_phi0=-100
hand_ampl=50

.....

print "Arm and Hand ready, start walking"
while walkTime >0 :
    x=rotation_omega*walkTime
    xHand=math.radians(x+arm_hand_phase)
    xArm=math.radians(x)
    hand_phi= hand_phi0+math.sin(xHand)*hand_ampl
    arm_phi= arm_phi0+math.sin(xArm)*hand_ampl
    arm.setAngleDeg(arm_phi)
    hand.setAngleDeg(hand_phi-arm_phi) # no hand angle is vs. body
    time.sleep(stepDelay)
    walkTime=walkTime - stepDelay

hand.fullStop()  # must be executed to free servos
```

Listing 3: Excerpts from **smoothWalk.py** python code implementing two correlated harmonic oscillator functions to encode more flexible and more smooth motion. The forward/backward direction is controlled by the relative hand-arm phase.

3.3 State Machine-based walking encoding, SC-188 approach

reflexMain.py is the discretized state-machine based interface to the Markov Crawler . It includes GUI allowing for execution of individual steps as well as of a pre-defined policy. The animation of the computed Crawler pose and readout of its measured position are displayed as well.

3.3.1 Defining physical dimensions of Markov Crawler

Before we launch the **reflexMain.py** we need to verify the dimensions of the physical Crawler you have build and the name of all configuration files are correct. Those are all config files required by **reflexMain.py** .

```
$ ls ~/markov-crawler/setup
arm.servo.conf generic.servo.conf  mouse.conf        walkForward.conf
crawler5.conf hand.servo.conf     walkBackward.conf
```

Please open in an editor the **crawler5.conf** master configuration file (shown in Listing. **??**) and verify the values are correct for the Crawler you have assembled. Refer to Fig. **??** for meaning of Crawler dimensions.

```
global crawlerConfig
crawlerConfig={}
crawlerConfig['name']="crawler5"
crawlerConfig['setupPath']="../setup/"

crawlerConfig['bodyLenght']=160 # unit (mm)
crawlerConfig['bodyHeight']=73 # unit (mm)

crawlerConfig['handLength']=77 # unit (mm)
crawlerConfig['handHighAngle']=math.radians(-40)  # unit (deg)
crawlerConfig['handLowAngle']=math.radians(-140) # unit (deg)
crawlerConfig['handNbucket']=5 # unit (state count)
crawlerConfig['handServoConfig']="hand.servo.conf"

crawlerConfig['armLength']=90 # unit (mm)
crawlerConfig['armHighAngle']=math.radians(39.1) # unit (deg)
crawlerConfig['armLowAngle']=math.radians(-30) # unit (deg)
crawlerConfig['armNbucket']=5 # unit (state count)
crawlerConfig['armServoConfig']="arm.servo.conf"

crawlerConfig['mouseConfig']='mouse.conf'
crawlerConfig['reflexForwPolicy']='walkForward.conf'
crawlerConfig['reflexBackwPolicy']='walkBackward.conf'

crawlerConfig['canvasLenght']=900 # unit pixels on the screen

# needed only for q-learning
crawlerConfig['learnDelay']=0.4 # additional delay when learning
```

Listing 4: The master configuration **crawler5.conf** describes dimensions of the physical Crawler and location of various calibration pickles.

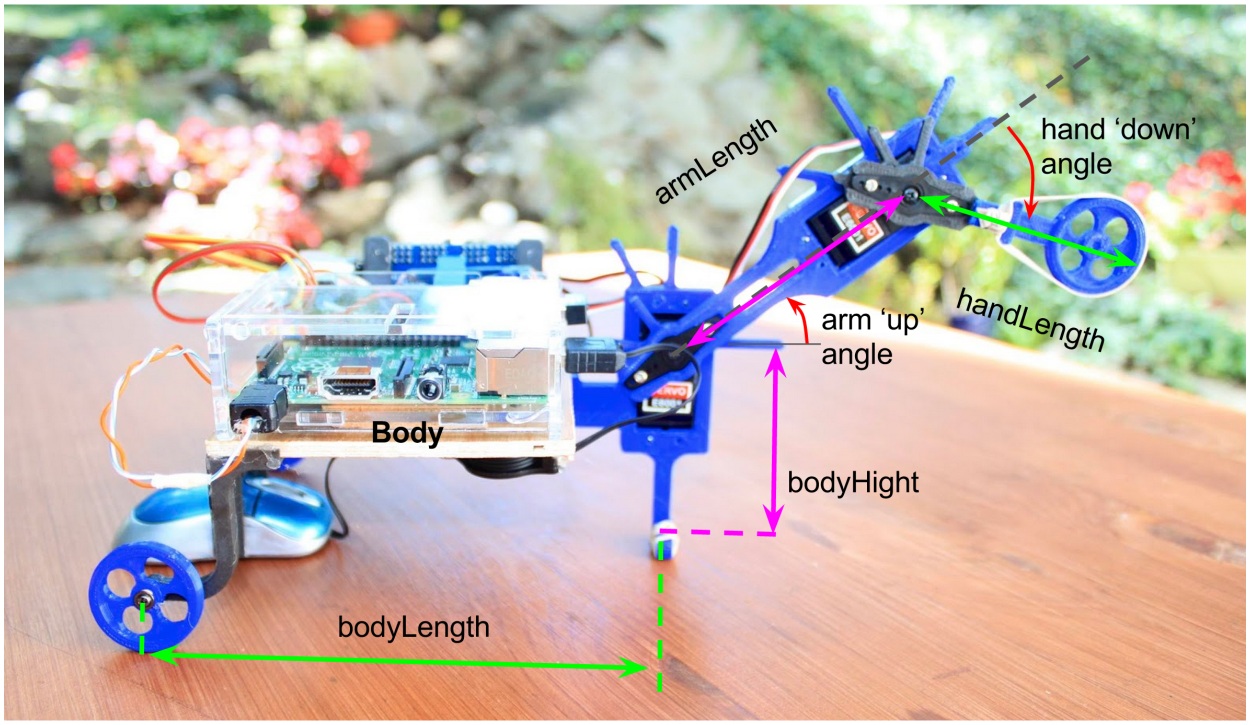


Figure 6: Measurement of Crawler dimensions should be done as indicated on this figure.

4 Calibration of mini-mouse

The reflex agents you executed so far did not relay on any feedback about Crawler movement. But Markov Decision Process needs to know it to generate rewards based the learning process. Therefore, we need to calibrate mini-mouse attached to the bottom of the Markov Crawler .

The software assumes the mini-mouse is attached to the port **/dev/input/mouse0**. To be 100% this is the case

1. unplug (all) USB mouse(s) from R-Pi
2. execute command below, you should see this answer

   ```
    $ ll /dev/input/mouse0
   ls: cannot access /dev/input/mouse0: No such file or directory
   ```

3. plug-in mini USB mouse
4. execute 'ls -l' again. Now you should see 'mouse0' and the current date

   ```
    $ ll /dev/input/mouse0
   crw-rw---T 1 root input 13, 32 Oct 11 19:01 /dev/input/mouse0
   ```

Now your mouse is visible by the R-Pi , we can calibrate its response. Each time the mouse moves we can ask it how far it moved - mouse will return an integer proportional to the traveled distance. We need to find a scale factor to convert this integer to mm. A simple program **mouseCalibMain.py** will help to find this constant.

First you need to prepare a setup: tape a sheet of paper to the table and draw 3 parallel lines, spaced by 10 cm. Place Markov Crawler with its foot on the middle line. Next, execute the calibration program (you do not need gksu). The calibration factor should be close to 0.020, so start with this value. Move Crawler forth and back by 10cm and observe if the reported traveled distance is about 100mm, as shown below. Once it does press 's' to save the calibration. or press 'q' to quit and start over with a slightly different constant.

```
 $ cd ~/markov-crawler/mouseUtil
 $ ./mouseCalibMain.py -c 0.020
 mouse dev=/dev/input/mouse0  scaleFactor=0.020000
 press: ENTER for distance or, s (save pickel) , q (quite w/o saving):
 mouse distance(mm) step=100.7
 mouse distance(mm) step=100.9
 mouse distance(mm) step=-102.9
 mouse distance(mm) step=-93.8
 mouse distance(mm) step=103.6
 mouse distance(mm) step=99.8
 press: s
saved pickel: ../setup/mouse.conf
```

After this process you should see newly created mouse calibration file:

```
$ ls -l  ../setup1/mouse1.conf
-rw-r--r-- 1 pi pi 75 Oct 11 21:40 ../setup1/mouse.conf
```

5 State Machine based reflex agent

At this point Markov Crawler is fully calibrated. Make sure the following setup files exist

```
$ ls  ~/markov-crawler/setup/
arm.servo.conf   mouse.conf         walkForward.conf
crawler5.conf   hand.servo.conf     walkBackward.conf
```

A Fresh copy of code from Bitbucket Git

If the **/home/pi/markov-crawler/ directory does not exist (or was erased by accident) you can always pull a fresh copy from the Bitbucket Git repository by executing this command after you ssh to R-Pi :**

```
$ cd
$ rm -rf markov-crawler
$ git clone https://balewski@bitbucket.org/balewski/markov-crawler
Cloning into 'markov-crawler'...
remote: Counting objects: 3, done.
.....
Unpacking objects: 100% (3/3), done.
```

B Full list of options for provided programs

B.1 servoCalibMain.py

```
$ cd ~/markov-crawler/servosUtil/
 ./servoCalibMain.py -h
Usage: servoCalibMain.py [options]
Options:
  -h, --help            show this help message and exit
  -m, --mockServos      disable servos
  -v, --verbose         print aux info
  -n NAME, --name=NAME  servo setup (pickle) name
  -s SETUP, --setupPath=SETUP  hardware configuration location
```

B.2 mouseCalibMain.py

```
$ cd ~/markov-crawler/mouseUtil/
pi@raspberry2 ~/markov-crawler/mouseUtil $ ./mouseCalibMain.py -h
Usage: mouseCalibMain.py [options]
Options:
  -h, --help            show this help message and exit
  -m MOUSE_DEV, --mouseDevice=MOUSE_DEV   mouse device
  -c PIX2MM, --conversionFactor=PIX2MM   pixel to mm conversion factor
  -n NAME, --name=NAME    mouse setup (pickle) name
  -s SETUP, --setupPath=SETUP    hardware configuration location
```