# CoSMo 2014 Lab - Cisek

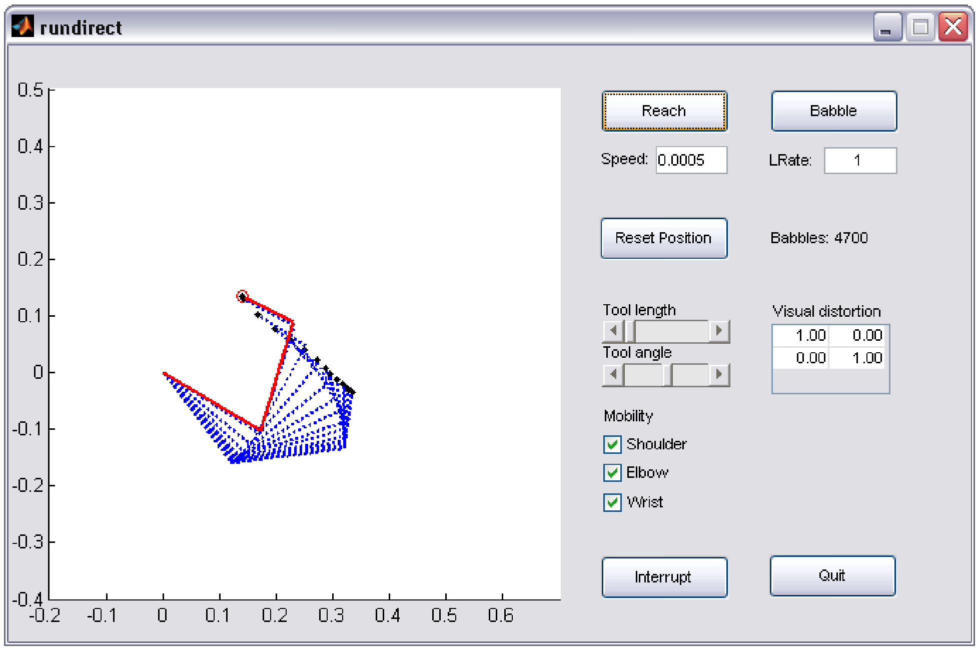
# The DIRECT model

## Overview

In this lab, we will simulate the DIRECT model of Bullock, Grossberg, & Guenther (1993). The simulation environment consists of the following MATLAB functions:

| | |
|---|---|
| initdirect: | This function initializes the model. By default, it creates a naïve, untrained model, which is returned in a large structure. You can also create a pre-trained model by calling initdirect(1). |
| rundirect: | This function runs the model created by initdirect. It opens up a small graphical user interface (GUI) window that contains a display of the workspace and the arm and some buttons (see below). The file rundirect.fig contains the MATLAB definition for the GUI. If you want, you can edit the figure using the MATLAB guide command. |
| direct: | This function implements the DIRECT model. By default, it performs a single movement to a given target in space. This is called by rundirect, so you don't have to call it yourself, but if you want to see how, type help direct. |
| jtox, xtov: | These functions implement the coordinate systems used by the model. |
| drawlimb: | This draws the current state of the limb on the current axes. |

Most of the functionality of the model is implemented within the direct.m file, which you should examine to understand how it works. You should also look at initdirect.m because it defines a lot of the variables that you might want to change if you choose to modify the model. The rundirect.m file implements the user interface, which might be very confusing if you're not familiar with MATLAB GUI philosophy, callback functions, etc. To make it a bit easier to read, I've put the most relevant functions at the top. These are the functions which set-up the window, execute reaching movements when you click in the workspace display, and produce motor babbling movements for learning. The rest of the file you can safely ignore.



Speed: 0.0005

LRate: 1

Babbles: 4700

Visual distortion

| | |
|---|---|
| 1.00 | 0.00 |
| 0.00 | 1.00 |

[✓ Shoulder]
[✓ Elbow]
[✓ Wrist]

## Exercise 1: Training the naïve model

First, initialize a model with a completely random inverse Jacobian matrix by typing:

```
model = initdirect;
```

The variable `model` contains a description of the model in a MATLAB structure. Now, open the interface by typing:

```
rundirect(model);
```

Notice that the "LRate" parameter is zero. This means that currently, the model will not do any learning.

Now, select the "Reach" button and click on the axes window. Watch how the limb flails around hopelessly for a bit and then stops (it does a maximum of 100 steps before giving up). Click a few more times to get it to flail around more. If it gets stuck in some awkward position, click "Reset Position" to give it a little help.

Now, set the "LRate" parameter to one, and again click a few points in the axes window. Notice that the limb is still pretty bad, but does sometimes make a good movement. Note: When the model succeeds in reaching a target, the limb is drawn in red. Click some more to watch it improve slightly over time.

To accelerate the training, deselect the "Reach" button and instead select the "Babble" button. Now watch as the arm makes random movements from different positions. The red lines show the directions of 50 random movements from each position. Again, if it gets stuck, just click "Reset Position" to give it a hand. Let this continue until the "Babbles" counter gets to 10000 or so.

Now turn of the "Babble" button and select "Reach". Now click to give it some more targets. Note that it is now much better, and tends to make fairly straight movements… most of the time. Why does it make straight movements? Note also that it will still be pretty bad sometimes, especially at the edges of its workspace. Why does it have trouble at the edges?

Try a few points near the middle of its workspace. If you see the model make a curved movement, or one with a bump, try giving it alternating targets so that it goes back and forth between two points. Don't bother being overly precise. Notice that the path straightens over time. Why? (And here's a tougher question: Why is this *more* effective if you *don't* always click the same exact two points?)

Let the model train itself some more, until you reach about 20000 babbles. Remember to "Reset Position" occasionally if it gets stuck in awkward joint configurations. After you reach 20000 babbles, try making some point-to-point movements in different parts of the workspace. If you find the model has trouble in a given region, bring the arm there and the let it "Babble" for a few seconds to give it some practice in that region. Work on the trouble spots until it performs fairly well in various locations. (But don't forget: The constraints on the joints make some places unreachable, at least without tools…)

## Exercise 2: Reaching with a tool

Now that you've trained your model, let's see what it can do. First, we'll try reaching with a tool.

Adjust the "Tool length" and "Tool angle" sliders until you see a small black tool held in the hand. It doesn't matter what angle or length, but don't make it too long. Now click a few targets.

What is it about the model that makes reaching with a tool possible? Note that this works even if you set "LRate" to zero and prevent any learning.

Note that you can now reach targets that were previously out of the limb's reach. How can that be if you never visited those targets during training?

Tougher question: What are the conditions in which reaching with a tool is less accurate than reaching without one? What is the reason for this?

To put the tool away, just set “Tool length” to zero by dragging the slider all the way to the left.

## Exercise 3: Constraining a joint

Let’s suppose the wrist joint has been injured by all that flailing around. Un-check the check-box next to “Wrist”. This is equivalent to putting a cast on the wrist, immobilizing it at its current angle. Now make some movements. Notice that the model can still reach most places, but not all, and its movements are slightly curved.

What is it about the model that makes this possible? It works even if “LRate” is zero. Remember, the inverse Jacobian gives you a joint rotation vector for *all* joints, so it assumes that they can all move. But the wrist can’t and yet the model still manages to complete the reach (usually). How?

Try checking and un-checking various combinations of the joints to see how it copes.

## Exercise 4: Reaching with visual distortions

In “visuomotor rotation” experiments, subjects are asked to point using a cursor on a screen, but the cursor’s movement is not directly matched to the movement of their hand. For example, sometimes a rotation is imposed such that the cursor motion is 90° clockwise or counter-clockwise to the actual hand movement.

Let’s try a counter-clockwise rotation. Click the top-left entry in the “Visual distortion” matrix, and change the 1 to a 0. Now hit TAB and change the top-right entry to 1. Hit TAB again and change the bottom-left to -1 and change the bottom-right to 0. Now set “LRate” to zero, and try a few movements.

Note that the endpoint often “orbits” around the target. Why?

Now set “LRate” back to one and let it babble for a bit. As before, give it some targets in different parts of the workspace so that it practices broadly. After an additional 5000 babbles or so, do some reaching movements. Note that most of these are much straighter now.

Ok, now let’s remove the distortion. Change the “Visual distortion” matrix back to the identity matrix and set “LRate” to zero. Make a few movements. What is happening now? Why?

Set “LRate” to one and let it recover back to normal with another few thousand babbling movements.

## Exercise 5: Changing limb length

The “Interrupt” button brings you to the MATLAB command line in debug mode (inside the `rundirect.m` file). One of the variables in your MATLAB workspace is called “`model`”, and it is the structure that contains all of the information about the model you’re currently running. This means you can change any parameter (such as link lengths) and then return to the GUI window by typing “`dbcont`” or “`return`”.

For example, try setting “`model.L2 = 0.3`” and then type “`return`”. Note that this has changed the length of the forearm to be 50% longer than it was before. Make some reaching movements to see if the model adjusts to this modification. How long did it take? Again, what is it about the model that makes it able to adjust so quickly?

## Additional exercises

As you become familiar with these files, you can invent a few other things to try. Below, I give you a few suggestions. Some of these are things where I have no idea what the outcome will be, so be creative and see what you can come up with.

**Nonlinear muscles**: Take a look at the function at the very bottom of `initdirect.m`. This defines how joint rotation commands are actually turned into joint rotations. At present, the mapping is simply unity. However, you can modify this so that motion at different joints is scaled differently (e.g. favoring the wrist before the shoulder) or so that there is a non-linearity that reduces commands that bring joints toward their extremes. Try playing around with this to see if it gives the model any advantage.

**Perturbations**: Add some code that lets you push the limb around unpredictably. How does it cope? Make the perturbations consistent in certain parts of the workspace and let the model learn. Can it to learn to “anticipate” and correct for consistent perturbations?

**Blindfolded reaching**: Currently, I did not implement the Position-Position-Map described in the 1993 paper. Quite simply: I ran out of time before the summer school! This means that the model cannot possibly make reaching movements without continuous visual feedback. If vision is turned off, it has no way of knowing its current position, and thus no way of calculating the appropriate spatial Difference Vector. So one thing you can try, if you’re feeling ambitious, is to implement the Position-Position-Map (look for PPMsm in `direct.m` and `initdirect.m`) and give the system what is effectively a Forward Kinematics Model. Now add some code to allow for blindfolded reaching and see how it performs.

**Kinetics**: The DIRECT model is not designed to deal with any sort of kinetic variables. However, as a feedback system it should be quite robust to a variety of perturbations (see above). So try giving its “motor plant” some second order dynamics and see how it copes. Presumably, there will be oscillations, but will the system become totally unstable? Try combining this with an implementation of muscles that are non-linear or act as a low-pass-filter.