

# **The Masked Priming Toolbox**

**Manual V. 1.0**

**Current Contact Information:**

Developer:  
Dr Andrew D. Wilson  
Psychology  
School of Social Sciences  
Leeds Beckett University  
Leeds UK

Primary email: [DrAndrewDWilson@gmail.com](mailto:DrAndrewDWilson@gmail.com)

Secondary email: [a.d.wilson@leedsbeckett.ac.uk](mailto:a.d.wilson@leedsbeckett.ac.uk)

Bug reports, feature suggestions, manual suggestions all welcome. The goal is to make this work in such a way that the only thing a user will ever need to do is alter the text parameter file.

*This software was developed with support by a grant from the Economic and Social Research Council ([www.esrc.ac.uk](http://www.esrc.ac.uk)) to Friederike Schlaghecken & James Tresilian.  
University of Warwick*

## Overview

The Masked Priming Toolbox is a collection of functions written in [Matlab](#) and incorporating the free third party [PsychToolbox 3 \(PTB3\)](#) that is designed to allow a researcher to run masked priming experiments using a variety of response devices. Very little knowledge of Matlab is required, as experiments are generated by creating a text file with the required parameters, and data is output to Matlab and Excel files for further analysis. You do need to know how to call a function in Matlab and use the help function.

The Toolbox is designed to present a variety of stimuli as primes, followed by one of several kinds of mask, and then a target that serves as an imperative stimulus requiring a response from a participant. These stimuli are generated prior to each experiment and are fully parameterisable; several common stimuli are coded and it is a straightforward matter to expand the set upon request.

The toolbox is made available to researchers under the following conditions:

1. Any publication of data collected using this toolbox should cite it. I intend there to be a short technical report published somewhere soon, but in the meantime please do something along the lines of:  
“We used the Masked Priming Toolbox for Matlab developed by Wilson, Tresilian & Schlaghecken (2011).”
2. The code is licensed under a GNU General Public License v3.0. Notes and additions to the manual also gratefully received.
3. The code is provided as is; development wise, this project is in beta. It has been tested extensively on the hardware noted later and works fine for us, however I strongly recommend you test it on your system.

The code is currently available via [Dr Friederike Schlaghecken's webpage](#) at the University of Warwick.

# General Notes

## *Software Requirements*

This is Matlab code written using Psychtoolbox 3. It has been tested up to MATLAB version 7.8.0 (R2009a) and up on the PC (WinXP mainly, some minor testing on Vista). It should be portable to the Mac but there may be subtle differences, in PTB behaviour especially, so you are advised to test it extensively. You will therefore need a copy of Matlab and you will need to install PTB-3.

**Matlab** is a software package designed for, primarily, engineering applications. It contains libraries (called ‘toolboxes’) of software for performing standard mathematical operations, graphing, modelling, etc. The basis of Matlab’s approach to data is linear algebra (the mathematical manipulation of matrices and vectors). Data is, in general, stored, accessed and manipulated in an  $m \times n$ -dimensional matrix form, with the location of a given piece of data being indexed by the coordinates of the matrix. Data is usually in a 1 dimensional matrix, called a vector, or a 2 dimensional matrix, and the convention for calling data is (row number, column number).

To get a good overview of the basics of Matlab, I recommend getting a copy of the following book (most other books are written for engineers):

Rosenbaum, D. A. (2007). *MATLAB for Behavioural Scientists*. Mahwah, New Jersey; Lawrence Erlbaum Associates.

**PsychToolbox (PTB)** is a suite of Matlab code written by psychologists to allow people to do psychophysical experiments using Matlab to control the stimuli, record and analyse data, etc. Matlab is not designed to handle graphics, etc, so PTB is based around a mex file called Screen. A mex file is just code written in another language (in this case, C and OpenGL) compiled in such a way that you can interact with it using Matlab. The practical upshot is that PTB does most of the hard work of graphics for you and you can control everything via Matlab, which is convenient because programming Matlab is easier than in C or OpenGL and you have access to huge libraries of pre-written code.

If you’re interested in using PTB yourself, visit these websites:

[Psychtoolbox Website](#)

[Online documentation](#)

[Psychtoolbox tech support forum](#) (requires Yahoo Groups login)

### **Installation:**

Go to [the installation page on the PTB home page](#) and follow the instructions precisely (I mean this – follow everything step by step and don’t skip anything). If it doesn’t work you are likely to have missed something so **uninstall completely** and start over. The installation entails installing a third party piece of software called Subversion that will handle the hard work, and then grabbing a Matlab .m file to run. Let PTB-3 install itself in the default location and *do not move it* – the latter is tempting because it won’t go into the normal Matlab toolbox folder, but that doesn’t matter. If you move anything it will not work.

You should run

```
>> UpdatePsychtoolbox
```

from the Matlab command window once a month, to get the latest bug fixes, patches, new code, etc. The computer will need to be online to be able to contact the server to download the updates, which will install themselves and tell you when it's done.

### ***Hardware Requirements***

See [the PTB hardware FAQ](#) for the latest information.

Pretty much any modern PC or Mac will be able to run this code without difficulty, although Matlab is a bit of a memory hog so the more RAM, the better (absolute minimum 512Mb, 1Gb preferred). You will need a decent graphics card capable of handling OpenGL commands (see below). These settings are all found in the 'Settings' tab of the desktop properties menu (right click on the desktop to get this option).

*Graphics cards:* The most common thing causing horrible looking errors is a graphics card that is not up to scratch. A graphics card is an additional chip with dedicated processing power and memory for handling low level graphics commands. Laptops are especially bad at this, as they tend to ship with useless Intel cards.

Right click on your desktop, and select 'Properties', then select the 'Settings' tab. You should see it say something like

'Display:

Plug and Play Monitor on NVIDIA GeForce 7300 LE'

The bit after 'Plug and Play Monitor on' is the name and type of graphics card installed. **If it says 'Intel' you need to upgrade:** Intel cards don't, as far as I can tell do anything at all, up to and including not having their own memory. **Any ATI Radeon or NVIDIA GeForce card should do the trick.**

If you ever start getting crazy errors about timing, etc, **check your graphics card, try UpdatePsychtoolbox, update your drivers for your graphics card** and then search [the PTB forum](#) for help. If none of those solutions work, *then* email me.

## ***Running Experiments***

Unzip the folder containing the Masked Priming Toolbox to a convenient location. Matlab has some problems with folders that have spaces, so by default I would put the folder in the C: directory, so that the path for the folder is C:\MaskedPrimingToolbox

Open Matlab and browse to the folder (using the Windows interface or by typing

```
>>cd C:\MaskedPrimingToolbox
```

at the command prompt (>>). You can set this to be the default location Matlab opens to by right clicking on the Matlab shortcut icon and adding the full path to the 'Starts in' box.

### **Parameter File**

You will need a parameter file set up for your experiment (see later for more detail) saved as a **tab-delimited plain text (\*.txt) file** to the main directory. The files **must** be in the format shown in the example Excel file – changing things will confuse the code resulting in an error from *readExperimentParameters.m*.

There are example parameter files saved as Excel documents with the code. The easiest way to create your own is to edit these in Excel and then Save As a tab-delimited text file.

Once you've made your parameter file, start an experiment by opening Matlab, changing to the correct directory, and typing

```
>> primingMain(fileName, subID);
```

where filename and subID are strings specifying the full filename and subject ID – see later.

**NOTE: You may need to disable antivirus software, etc – anything that might run in the background – so as to preserve timing, etc**

It is good practice to run a practice block with at least one of every trial type in it. Not only does this generally make sense, it means that Matlab can 'preload' the various required functions into memory, which cuts down on the time to draw things and helps ensure your timing is preserved.

# Running Experiments

## Using *test.m*

*test.m* is a function that will run through one full block of your experiment with randomisation turned off and with 2s pauses in between each display. You can escape out during any of these pauses. You should run through your displays to make sure everything is displaying the way you expect it to. Call it from the command window as

```
>> test(fileName);
```

## The Splash Screen

This is the screen that is shown at the start of an experiment and in between blocks. Pay attention to the parameters here and make sure everything is where you want it. Use *test.m* to check your screen.

## Button response tasks

Make sure you have coded your correct responses properly, i.e. using the right name for the key. Run *utilities('keyName')* from the Matlab command window to check the correct name.

## Tablet tasks

I have made every effort to catch any errors that will cause a crash; however there is an occasional error caused when I try to pull data from the tablet and there's none there that causes the experiment to end, and I don't currently have a reliable way to catch this error.

In amongst the output sent to the Matlab command window is information about how many blocks were completed and how many trials of the last block. Use this information to finish the experiment. If, for example, you had completed 3 blocks from a run of 10, open your parameter file, set *nBlocks* to 7, save this, and rerun *primingMain.m* with a new subject ID (e.g. subXXa for subject XX). You will then end up with two data files which can be put back together again by calling *compileSessions(subID1, subID2)* – see *help compileSessions* for more detail.

## Force tasks

The most common error will arise if you have the transducer plugged into the wrong USB port. This is currently set to 'COM3' in the Matlab code, but should probably become a parameter at some point. If you get errors from the PTB function *IOPort*, this will be the cause.

## Timing Errors

If your computer is having timing errors, PTB will return various warnings, e.g.

```
>> INFO: PTB's Screen('Flip') command seems to have missed the  
        requested stimulus presentation deadline  
>> INFO: a total of 3 times during this session.
```

A few missed 'Flips' is not really a problem, however a lot of missed 'Flips' (more than, say, 10) suggests real timing problems.

### Solutions:

#### 1. Run

```
>> refreshRate = Screen('NominalFrameRate', 0)
```

from the Matlab command window. This is the command that pulls the refresh rate out automatically to scale the stimuli durations – if it is not reporting a number that makes sense (e.g. not an integer, not the refresh rate you specified via your OS) you have difficulties.

#### 2. Update

a. Psychtoolbox

b. your graphics card drivers

#### 3. Search the [Psychtoolbox forum](#) for help.

### General things

1. Your practice block should consist of at least one full block of your experiment (i.e. use a copy of your parameter file with **nBlocks** set to 1). The main thing this does is use everything once, and this serves to preload all the relevant Matlab and PTB function calls into memory. This helps ensure that all the timing, etc really is preserved.
2. Keep a copy of the parameter files that ran your experiment in a folder with the raw data. This means you can rerun a task precisely in the future to check stimuli, remind yourself of details, etc.
3. Place all the data in a single folder (I generally make a folder for every experiment and create a subfolder called 'Raw Data'). This is the folder path name you can then feed into *collateAllSubjects.m* to get all the data output to a single Excel file.
4. Pilot all experiments: generate one complete data set and check the output to ensure that the data is recording properly, and everything is labelled and sorted the way you want (especially making sure that your **TrialType** labels are correctly sorting the data; see the section on Trial Parameters). Do this especially when you are new to using this toolbox.



## primingMain.m

This is the function you will call to begin an experiment. It handles reading in the text parameter file and calling the right code to run your experiment.

You call this function from the Matlab command window as follows:

```
>> primingMain(fileName, subID);
```

fileName must be a string specifying the complete file name of the parameter file. subID is a string specifying a subject ID – I recommend ‘subXX’; e.g.

```
>> primingMain('ExperimentParameters.txt', 'sub01')
```

Note the single quotes around the strings; this is Matlab syntax and the text will only turn blue once you’ve closed the quotes properly.

In Matlab, you can call a previously used command by pressing the up-arrow key. When running multiple subjects, simply press the up-arrow and edit the subID to the next number.

**If you get errors, read them.** Matlab has useful error output and I’ve added a lot of error messages as well.

*Common problems:*

Incorrectly set up parameter file, resulting in errors labelled as coming from *readExperimentParameters.m*.

Horrible timing errors (text generated by the Psychtoolbox) – see above for solutions.

## ExperimentParameters.txt

This **tab-delimited** text file is the primary user interface and contains all the parameters required to run a priming experiment. These must be set correctly and any run-time errors you encounter are highly likely to be caused by you not doing this right.

Open the parameter file in Excel to edit. Cell numbers (e.g. A1) refer to the cell location in Excel. Once done, save the file as a **tab-delimited (.txt.) text file** (File > Save As). Saving in any other format will make *readExperimentParameters.m* fail (producing errors in the Matlab command window about failing to read parameters). Excel will mention a couple of limitations of this file format – just click ok to them all.

### Experiment Parameters

**ExperimentParameters** (A1) is just a string labelling the file that gets skipped when read into Matlab – there is no need to change, although if you're feeling compulsive this is a good place to put the full name of the specific parameter file you're working with.

Row 2 contains labels for the parameters. Again this gets skipped when read into Matlab but do not change these or else you will lose track of what your columns mean.

Row 3 contains values for the parameters labelled in Row 2:

**ResponseDevice** takes *tablet* or *button* or *force* as arguments.

**Masking** takes *unmasked* or one of *'lines'*, *'randomLines'* or *'squares'* to specify whether to mask the prime and if so, how. See *help makeMask* for more information.

**MaskTargetISI** takes *yes* or *no* to specify whether the mask and target should be simultaneous or not. If 'yes', make sure you set their locations to be different or else they will simply get drawn on top of each other in the single frame that displays them.

**Randomisation** takes *yes* or *no* to specify whether the trials should be randomised or run in the specified order.

**TabletStartLocation** takes one of the strings *centre*, *left*, *right*, *topCentre*, *bottomCentre*, *topLeft*, *bottomLeft*, *topRight*, or *bottomRight* or a set of coordinates (see below) to specify where on the tablet the start location should be placed. This is the location on the tablet to where the stylus must be returned to trigger the next trial in a tablet experiment. It's not used in the button experiment but should remain specified so that *readExperimentParameters.m* isn't tripped up by the missing field.

You can also specify a start location directly by entering the (x,y) coordinates where you want it centred. The location **MUST** be specified as

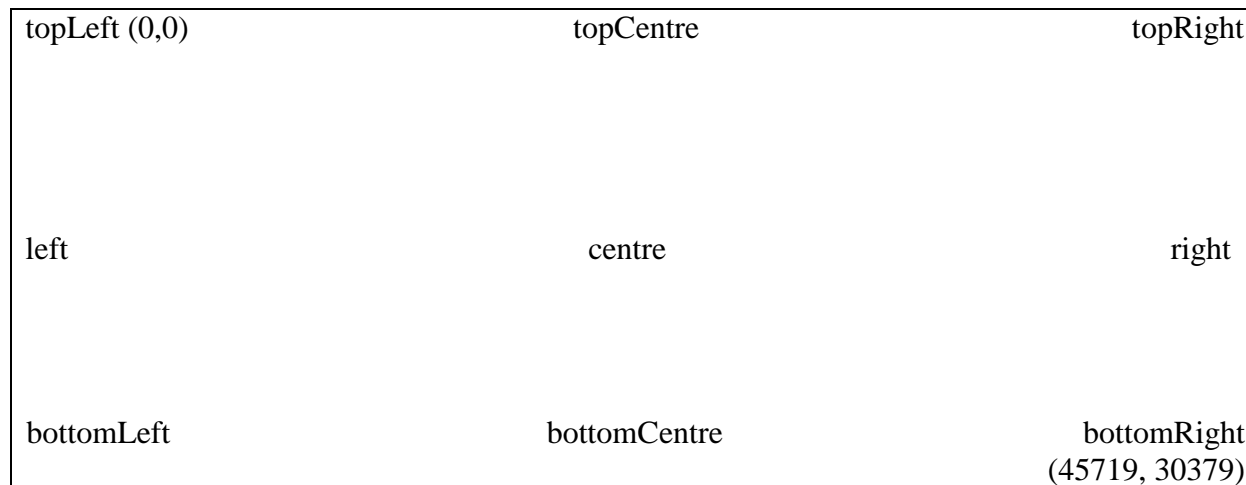
[x,y]

e.g.

[1024,768]

Note the square brackets, the comma and no whitespace.

Settings for an A0 Wacom intuos<sup>2</sup> tablet:



**nBlocks** is the number of blocks you want

**nRepetitionsPerTrialType** is the number of times within a block you want the unique trials types specified below repeated. One block therefore has  $n\text{UniqueTrialTypes} * n\text{RepetitionsPerTrialType}$  in it – it's arranged this way so that randomisation is across trial types but within blocks.

**TrialLength** takes the string *fixed* or *variable*: *fixed* trial length means that the trial runs for the time it takes to display the stimuli + **ResponseDuration**, while *variable* trial length is event based (the trial waits until a response is made, then waits for **ResponseDuration**, then starts the next trial). Note *variable* is only implemented for *button* tasks, because those are currently the only ones that contain a

way to define a trial ending event other than time. As a rule of thumb you want movement tasks to just run for a set amount of time anyway (piloted for various populations).

**ResponseDuration** is the amount of time, in milliseconds, that you want to collect data for after stimulus presentation is done (for **TrialLength** = *fixed*; typical values are 1300ms for the button, 1500ms for the tablet and 3000ms for the force transducer) or the amount of time to wait after a response is made before starting the next trial (for **TrialLength** = *variable*).

## Mask Parameters

Masks are randomly generated, one per trial, using a specified number and type of elements of random length and width. These characteristics are all parameterisable and should be tailored to mask your stimuli.

The basic algorithm is

1. the mask of size (**xExtentPixels**, **yExtentPixels**) is subdivided into boxes of size **boxSize**. The division does not have to be even, i.e. you can have a mask of size (100, 100) and boxes of size 11, or whatever.
2. each box has an element placed randomly within it. These elements can be lines (using 'lines' or 'randomLines') or squares (using 'squares'). More element types can be implemented on request.
3. each element is randomly sized according to the mean and standard deviations specified in those parameters. Setting the SD to 0 means no variation.

Your mask should be big enough to cover your prime, and you vary the density by altering boxSize and the element lengths (bigger box sizes means fewer boxes, and so on).

**MaskParameters** (A4) is just a string labelling the file that gets skipped when read into Matlab.

Row 5 contains labels for the parameters. Again this gets skipped when read into Matlab but do not change these.

Row 6 contains values for the parameters labelled in Row 5:

**xExtentPixels** is the width, in pixels, of the mask

**yExtentPixels** is the height, in pixels, of the mask

**boxSize** is the size, in pixels, of the boxes that the mask is subdivided into. Each box gets an element, so varying these first three parameters varies the size and density of the mask.

**meanPenWidth**, **penWidthSD** are the mean and standard deviation, in pixels, of the normal distribution from which each line width is drawn

**meanLineLength**, **lineLengthSD** are the mean and standard deviation, in pixels, of the normal distribution from which each line length is drawn

## Splash Screen Parameters

The Splash Screen is the screen that comes on at the start of the experiment and at the start of each block to remind participants of the targets they will see (or primes they may or may not see, in the case of a primeID experiment) and the required response. This section takes a variable number of imperative stimuli, one per **Response**, which must be parameterised, one per row, between the line of labels and the row that says ‘EndSplashScreen’; **this last line must not be altered or readExperimentParameters.m will fail.**

This code is premised on the fact that there is only ever one response per imperative stimulus. For each **Response**, the code looks for a trial that takes that response and draws the corresponding imperative stimulus (target or prime) in the specified location and in the specified location, along with the correct response for that stimulus. To scale your stimuli locations, check your screen resolution or call *utilities('screenSize')* to tell you what it is. See *help splashScreen* for more information.

**SplashScreenParameters** (A7) is just a string labelling what’s coming next that gets skipped when read into Matlab.

Row 8 contains labels for the parameters. Again this gets skipped when read into Matlab but do not change these.

Row 9 and on contains values for the parameters labelled in Row 8 – each row corresponds to a different imperative stimulus that is to be drawn onto the splash screen

**ImperativeStimulus** – takes a string specifying ‘prime’ or ‘target’. In general the imperative stimulus for an experiment will be the target; if you are running a prime identification experiment, however, the imperative stimulus will be the prime. In this later case the experiment should not display any targets (i.e. **Target** will be set to ‘none’).

**StimulusType** – takes a string that describes the stimuli participants will need to respond to. As per **Trial Parameters**, this takes anything *makeStimuli.m* can make and should obviously match the imperative stimulus.

**Orientation** – a number specifying the imperative stimulus orientation (e.g. a left pointing arrow takes 0, a right pointing arrow takes 180, and so on – see below for details)

**Response** – a string specifying the required response, which will be written to the screen and must match that for the imperative stimulus.

**Location** – string specifying the coordinates where on the screen you want this example positioned; currently requires the coordinates in the form ‘[x,y]’, but I will eventually code up some default locations.

## Trial Parameters

These are the parameters for the specific trials. You need one line for each unique trial type, e.g. if your experiment has factors Compatibility (Compatible, Neutral, Incompatible), Target Direction (Left, Right), and Target Location (Above/Below the mask) then you would need  $3 \times 2 \times 2 = 12$  unique trials to be specified. You must include a specification for everything, even factors you would normally eventually average over such as Target Location. If you want to group trials (i.e. specify that as far as you're concerned, Compatible/Left/Up is the same as Compatible/Left/Down) give them the same label in **TrialType** – see below.

**TrialParameters** (A12 or so) is just a string labelling the file that gets skipped when read into Matlab.

Row 12 (or so, depending on how many splash screen stimuli you've specified) contains labels for the parameters. Again this gets skipped when read into Matlab but do not change these.

Row 13 (or so) and on contains values for the parameters labelled:

**UniqueTrialNumber** is a column of consecutive numbers that runs from 1 – nUniqueTrials. You can use this later to identify specific trials, etc, so make sure that it does run consecutively.

**TrialName** is a unique name for each trial. This column is mostly for you, and I suggest you give it a name that lists which levels of all your factors this trial represents (e.g. CompatibleLeftUp, NeutralRightDown). These can be as long, etc as you like but should contain no whitespace and should uniquely describe the trial.

**TrialType** is a way for you to group trials for analysis. When *sortData.m* runs and computes median RTs, it collates all trials labelled as a given **TrialType** and finds their median. This column will mostly contain repeats, therefore (e.g. 'Compatible', 'Incompatible'). Make this column as useful and informative as possible – label things in such a way that if you sort data by this column in Excel, the data is ordered the way you might like to have it (NB: sorting routines tend to treat 'trialType2' as coming after 'trialType10', i.e. they do a nested sort by first number, then second. If you have double digits and single digit numbers, pad the single digit numbers out with 0s, i.e. 'trialType02').

**Feedback** specifies whether or not you want feedback for this trial (takes 0 for no, 1 for yes). This currently only works for button presses and force transducer experiments because the tablet responses don't get scored till later.

a. **Button feedback** is text that reports 'Correct!' or 'Incorrect!'

b. **Tablet:** not implemented yet

- c. **Force transducer** feedback consists of an animated bar that represents the current force being exerted as a function of the measured maximum, and a target oval that is centred at the target proportion of maximum voluntary contraction specified under **CorrectResponse** (see **CorrectResponse** below).

NB: if you turn feedback on for the force transducer, the effective sampling rate drops from 100Hz to the screen refresh rate because the transducer does not buffer and the feedback is real-time (to allow training).

The next parameters specify the prime:

**Prime** is the stimulus you want to use as prime. The options are currently *doubleArrow*, *brackets* and *tripleLines* (typical neutral primes), *cross* (makes a square, symmetrical cross), *exCross* and *exSquare* (makes an ‘exploded’ cross/square, i.e. four line segments in which the lines don’t touch) *crossXX* (e.g. ‘cross20’, where XX = the proportion of the way up the vertical line you want the horizontal line; net result is a rectangular cross with a long vertical axis and the horizontal line, e.g. 20% of the way up), *rectangle*, as well as *none* (which will simply flip to a blank screen but preserve everything else, including timing) and will eventually include *singleArrow* plus anything else that is required.

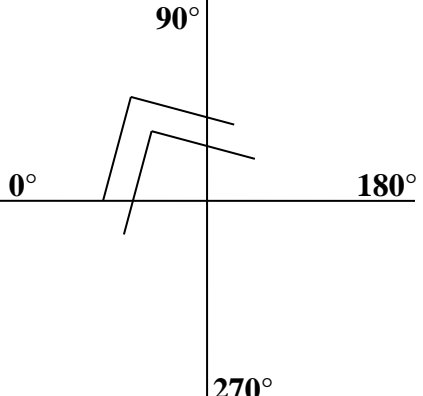
**PrimeDuration** is the number of frames for which the prime should be displayed. This should reflect your screen refresh rate (e.g. on a 60Hz monitor one frame = 16.67ms).

**PrimeLocation** is where on the screen you want the prime to be centred, relative to the mask. Valid locations are *centre*, *left*, *right*, *up*, and *down*. Everything that isn’t ‘*centre*’ places the middle of the prime outside the mask in the direction specified. Alternatively you can directly specify a location, as per **TabletStartLocation** (i.e. the same rules – no whitespace, square brackets, etc: e.g. [1024,768]).

**PrimeSize** is a gain on the size of the stimulus, i.e. the default values for a prime size are all multiplied by this value. Values between 0-1 therefore produce smaller primes, values greater than 1 produce larger primes. **Default prime size is approximately 65x40 pixels.** This will likely get replaced at some point by a more precise method.

**PrimeOrientation** = the direction you wish the prime to point, in degrees. This frame of reference (0° = left, increasing = clockwise rotation; see figure) is standard for graphics.



	<p>Frame of reference, with an example arrow shown at 60°. Arrows default (with PrimeOrientation set to 0) to left pointing (i.e. &lt;&lt;). A right pointing arrow therefore has PrimeOrientation = 180 (to produce &gt;&gt;), and so on.</p> <p>Brackets as stimuli, with PrimeOrientation set to 0, are upright (i.e. []).</p>
---	---

The next parameters specify the mask and blank duration:

**MaskDuration** is the number of frames for which the mask should be displayed. This should reflect your screen refresh rate (e.g. on a 60Hz monitor one frame = 16.67ms).

**MaskLocation** is where on the screen you want the mask to be, relative to the centre of the screen. Valid locations are *centre*, *left*, *right*, *up*, and *down*. Everything that isn't 'centre' places the middle of the target one mask width or height away from the centre of the screen. Alternatively you can directly specify a location, as per **TabletStartLocation** (i.e. the same rules – no whitespace, square brackets, etc: e.g. [1024,768]).

**BlankDuration** is the number of frames for which the blank between the mask and target should be displayed. This should reflect your screen refresh rate (e.g. on a 60Hz monitor one frame = 16.67ms).

The next parameters specify the target and are identical in nature to the prime parameters described above:

**Target** is the stimulus you want to use as target. The options are currently *doubleArrow*, *cross* (makes a square, symmetrical cross), *exCross* and *exSquare* (makes an 'exploded' cross/square, i.e. four line segments in which the lines don't touch) *crossXX* (e.g. 'cross20', where XX = the proportion of the way up the vertical line you want the horizontal line; net result is a rectangular cross with a long vertical axis and the horizontal line, e.g. 20% of the way up), *rectangle*, as well as *none* (which will simply flip to a blank screen but preserve everything else, including timing) and will eventually include *singleArrow* plus anything else that is required.

**TargetDuration** is the number of frames for which the target should be displayed. This should reflect your screen refresh rate (e.g. on a 60Hz monitor one frame = 16.67ms).

**TargetLocation** is where on the screen you want the target to be, relative to the centre of the screen. Valid locations are *centre*, *left*, *right*, *up*, and *down*. Everything that isn't '*centre*' places the middle of the target one mask width or height away from the centre of the screen. Alternatively you can directly specify a location, as per **TabletStartLocation** (i.e. the same rules – no whitespace, square brackets, etc: e.g. [1024,768]).

**TargetSize** is a gain on the size of the stimulus, i.e. the default values for a target size are all multiplied by this value. Values between 0-1 therefore produce smaller targets, values greater than 1 produce larger targets. **Default prime size is approximately 65x40 pixels.** This will likely get replaced at some point by a more precise method.

**TargetOrientation** = the direction you wish the target to point, in degrees. This frame of reference (0° = left, increasing = clockwise rotation; see figure above) is standard for graphics.

The final column is the correct response

**CorrectResponse** is a string detailing the correct response. It is important that this gets coded correctly because it's used to score the responses.

- a. **Keyboards:** enter the name of the key corresponding to the correct response. To check the name, run  
`>> utilities('keyName')`  
from the command line in Matlab and follow the instructions.
- b. **Tablet:** Not currently implemented until this gets scored on a trial by trial basis.
- c. **Force Transducer:** requires a number between 0 and 1 that is the target proportion of maximum voluntary contraction for that trial. This gets used to scale the feedback display (see **Feedback** above).

## Experiment Code Help Files

The following is the text from the Matlab code running the experiments. Fully up to date versions of this help text is always available by typing 'help functionName' in the Matlab command window (assuming you are in the same directory as the functions).

In alphabetical order:

- analyseActionData.m
- analyseKinematics.m
- collateAllSubjects.m
- compileSessions.m
- kinematicLandmarks.m
- makeMask.m
- makeStimuli.m
- maskedPrimeForceMain.m, maskedPrimeMain.m, maskedPrimeTabletMain.m
- primingMain.m
- readExperimentParameters.m
- sortData.m
- splashScreen.m
- test.m
- unmaskedPrimeForceMain.m, unmaskedPrimeMain.m,  
unmaskedPrimeTabletMain.m
- utilities.m

```

function analyseActionData(subID, exptType)
% function analyseActionData(subID, exptType)
%
% This is the primary hub for the analysis of kinematic data from the
% tablet. It handles all the output to Excel of the data sorted by
% condition, as specified in the cell array experimentData, after
% sending the data to analyseKinematics.m
%
% ARGUMENTS
% subID:      string specifying the subject ID
% exptType:   string specifying 'masked' or 'unmasked' (this matters
%             for computing RT as movementOnset -
%             startOfStimuliPresentation)
%
% OUTPUT
% This code saves out a collection of files, all prefixed 'subID_'
%   subID_sortedData:
%       x_Raw           Raw data
%       y_Raw
%       x_ZerosFixed    Raw data with 0s (indexing an empty
%                       pkt) replaced with the data point
%                       from time(pkt=empty)-1
%       y_ZerosFixed
%       correctionCountX A count of how often this happened
%       correctionCountY
%       onsets
%       reactionTimes    (NB this gets exported to an Excel
%                       file called
%                       'subID_ReactionTimes.xls')
%   + the corrected position time series sorted by trialType
%       Compatible_Left_X   Compatible_Left_Y
%       Compatible_Right_X  Compatible_Right_Y
%       Incompatible_Left_X Incompatible_Left_Y
%       Incompatible_Right_X Incompatible_Right_Y
%       Neutral_Left_X      Neutral_Left_Y
%       Neutral_Right_X     Neutral_Right_Y
%
% SUB-FUNCTIONS
% xlsheets(sheetnames,varargin) - used to create custom workbooks,
% code from the Mathworks website

```

```

function [experimentRecord onsets dataUsed] = analyseKinematics
(experimentRecord, subID, displayTime, rawDataX, rawDataY)
% function [experimentRecord onsets dataUsed] = analyseKinematics...
%         (experimentRecord, subID, displayTime, rawDataX, rawDataY)
%
% Filters (via a dual pass filter with Butterworth 'max flat'
% configuration parameters) and differentiates (via dx.m) kinematic
% data from a masked priming study. It then calls kinematicLandmarks.m
% to establish the various features of the resulting speed profile and
% computes RT. The kinematic landmarks plus RT are appended to experimentRecord.
%
% NB RTs < 100ms are scored as 'errors'. Negative RTs index trials in
% which there was enough movement before the trial began to trip the
% onset calculation; these kinematics should be eyeballed
%
% ARGUMENTS
% experimentRecord: cell array storing all the trial information for
%                  the rawDataX and Y files
% subID:           string specifying the subject ID
% displayTime:     time (in s) between primeOnset and targetOnset
%                  that must be subtracted off the movement onset to get RT
% rawDataX:        matrix of X position data (rows = time, columns =
%                  trials)
% rawDataY:        (optional) matrix of Y position data (rows = time,
%                  columns = trials)
%
% RETURN VALUES
% experimentRecord: updated cell array with RT, MT, TPS, DT and PS
%                  appended
% onsets:           The indices of the vector where movement onset was
%                  identified.
% dataUsed:         The data on which everything was computed (either
%                  X or XY data)
%
% This code also outputs Matlab files called 'subID_kinematics.mat'
% that contains
%     experimentRecord
%     rawDataX
%     filteredX
%     velocityX
%     onsetTimeX
%     xAverage (this only makes sense if the data passed is sorted
%               by trial type)
%     xSD      (this only makes sense if the data passed is sorted
%               by trial type)
%     reactionTimes
% + (if the data is 2 dimensional)
%     rawDataY
%     filteredY
%     velocityY
%     onsetTimeY
%     yAverage (this only makes sense if the data passed is sorted
%               by trial type)
%     ySD      (this only makes sense if the data passed is sorted
%               by trial type)
%     rawTangentialVxy
%     velocityXY
%     onsetTimeXY
%
% SUBFUNCTIONS
% dx = dx( x, fs ) - differentiation routine

```

```

function dataFileNames = collateAllSubjects(pathName)
% function collateAllSubjects(highestSubjectNumber, exptType)
%
% Scans through all the .mat files stored in a given folder pathName,
% loads them, and collates the data found in the cell array 'medians'
% across subjects. It then outputs the result into an Excel file at
% pathName called 'Collated Data', with three sheets (Median RTs,
% Errors, and Misses). NB: a common error with old data will be a
% failure to collate 'misses' which were previously treated as errors.
% Contact Andrew with your Matlab data file and I'll reformat it to
% work. This should not be a problem for future experiments.
%
% The function figures out the subjectID by scanning the file name of
% the current .mat file until it hits an underscore - this matches the
% convention that all data files generated by the Masked Priming
% Toolbox are of the form subID_fileName. If you alter this convention
% for any reason this function will fail.
%
% To add: compute mean and SD across subjects
%
% ARGUMENTS
% pathName: a string specifying the complete path of the folder where
%           the data is.
%
% Andrew D Wilson 2009

```

```
function compileSessions(subID1, subID2)
% function compileSessions(subID1, subID2)
%
% Compiles together two parts of data from a single subject; copes with
% the times when the tablet experiments crash out. Data is saved out to
% a file called 'subID1_maskedPrimeTabletDataREMOVETHIS.mat'; good
% practice is to move the original files into a separate folder (ie do
% not throw them away) and to then remove the text that says
% REMOVETHIS. If for any reason it's not tablet data, rename the file
% to match the appropriate convention.
%
% ARGUMENTS
% subID1: the subID from the first run, usually subXX
% subID2: the subID from the second run, usually subXXa
%
% Andrew D Wilson 2009
```

```

function [onsetTimeB offsetTimeB timeToPeakSpeed peakSpeed movementTime
decelerationTime] = kinematicLandmarks(data, tolerancePercent)
% function [onsetTimeB offsetTimeB timeToPeakSpeed peakSpeed movementTime
%   decelerationTime] = kinematicLandmarks(data, tolerancePercent)
%
% Computes the vector indices in a position time series that the
% various key landmarks occur. It's up to the user to use these vector
% indices to figure out what the actual time is - this code is
% indifferent to sampling rate. This works happily on single peak
% speed data from a discrete movement (although in theory you could
% chain together a series of calls to this code, chopping a single
% time series from a series of discrete movements according to the
% computed movement on- and offsets).
%
% Implements Algorithms A & B from
% Teasdale N, Bard C, Fleury M, Young DE, Proteau L. (1993)
%   Determining movement onsets from temporal series. Journal of Motor
%   Behavior, 25(2):97-106
% to estimate movement onset and offset
%
% ARGUMENTS
% data: a 1-dimensional vector of speed time series data (ie the
%       result of filtering and differentiating position data) This
%       function is currently best suited to discrete movements with a
%       single peak speed. If you have noisy (multiple speed peaks)
%       data you should plot the output to double check what comes out
% tolerancePercent: (optional) percentage (eg .01 for 1%) parameter
%                   for the Teasdale et al algorithm. It defaults to 5% - this
%                   effectively means that any signal <5% of the peak speed gets
%                   filtered, which is ok. When first running an experiment it's
%                   probably a good idea to tweak the tolerancePercent to see what
%                   gets cut out, though.
%
% OUPUT
% onsetTimeB:      Data point in vector that satisfies the criteria
%                  for movement onset, as per Teasdale et al
%                  (Algorithm B)
% offsetTimeB:     Computed the same way as onsetB except starting
%                  from the end and working backwards
% peakSpeed:       max(abs(data)) so it doesn't worry about direction
%                  data (ie the sign). Use the indices to look up
%                  the data yourself
% timeToPeakSpeed: Data point at which PS occurs
% movementTime:    offsetTimeB - onsetTimeB
% decelerationTime: movementTime - timeToPeakSpeed
%
% Andrew D Wilson 2008

```



```

function masks = makeMask (nMasks, maskLocations, maskParameters)
%function masks = makeMask (nMasks, maskLocations, maskParameters)
%
% Creates nMask masks by drawing randomised elements to a grid. The
% grid size is specified by the variables xExtentPixels and
% yExtentPixels, which should be computed to produce a mask of
% approximately 1.5 x 1 degree of visual angle. This will depend on
% your screen's physical size and resolution and the size of the prime
% you want to mask.
%
% Mask specifications:
%   'lines':      creates a mask similar to ERTS, with 1/6 of the
%                 lines vertical, 1/6 horizontal and the remaining
%                 2/3s with random orientation
%   'randomLines': all line orientations randomly generated
%   'squares':    each mask element is a randomly sized square with
%                 random line width. This was made to try and mask
%                 squares and is otherwise not a lot of use, but
%                 I've kept the functionality
%
% ARGUMENTS:
% maskLocations  = array of mask locations specified as trial
%                 parameters
% nMasks         = a 1x2 vector [nTrialsPerBlock nBlocks]
% maskParameters = struct containing parameters from
%                 ExperimentParameters.txt
%
% RETURN VALUES:
% masks = a struct, containing fields
%   maskCoordinates = a 2xnMask*2 matrix containing the coordinates
%                     of all the lines, set to draw from (0,0) on
%                     the monitor. Compute sourceRect as
%                     [0 0 max(maskCoordinates{trial}(1,:) max(maskCoordinates{trial}(2,:))]
%                     Use these coordinates as arguments to a
%                     Screen('DrawLines') call.
%   penWidths      = nMasks x nLines matrix with randomly generated
%                     pen widths. Again for use with a call to
%                     Screen('DrawLines')
%   sourceRects     = nTrial x 4 matrix with the coordinates of the
%                     sourceRect for each trial's mask. Use this to
%                     figure out size, etc for positioning
%
% Andrew D Wilson (2009)

```

```

function stimuli = makeStimuli (maskSize, trialParameters)
% function stimuli = makeStimuli (maskSize, trialParameters)
%
% Creates two nTrialsx1 cell vectors of coordinates for drawing fully
% parameterised stimuli. There is a vector for primes and for targets.
%
% Each cell in the vector contains a 2x2*nLines matrix of coordinates
% suitable for use with Screen('DrawLines'). The matrix is arranged:
%     [ originX   endX1 ... originX   endXn
%       originY   endY1 ... originX   endYn]
% for n lines in the stimulus.
%
% ARGUMENTS
% maskSize:           the size of the largest random mask for this
%                     experiment - in the form [maxX maxY]
% trialParameters:    a struct array created when
%                     ExperimentParameters.txt is read in via
%                     readExperimentParameters.m
%
% RETURN VALUES:
% stimuli = a struct, containing fields
%     primeCoordinates: an nTrialx1 cell vector of coordinates for
%                     the prime for that trial
%     primeSourceRect:  an nTrialx4 matrix with the coordinates that
%                     specify the rect that the primes for that
%                     trial are drawn in
%     primeDestRect:    an nTrialx4 matrix with the coordinates that
%                     specify the rect that the primes for that
%                     trial are drawn in
%     targetCoordinates: an nTrialx1 cell vector of coordinates for
%                     the target for that trial
%     targetSourceRect: an nTrialx4 matrix with the coordinates that
%                     specify the rect that the targets for that
%                     trial are drawn in
%     targetDestRect:   an nTrialx4 matrix with the coordinates that
%                     specify the rect that the targets for that
%                     trial are drawn in
%
% SUBFUNCTIONS
% makeTripleLines
% makeSingleArrow
% makeDoubleArrow
% makeBrackets
% makeCross
% makeCross2
% makeSquare
% makeRectangle
%
% Andrew D Wilson (2009)

```

```

function [experimentRecord timeStamps] =
maskedPrimeForceMain(experimentParameters, maskParameters, splashParameters,
trialParameters)

function [experimentRecord timeStamps] = maskedPrimeMain(experimentParameters,
maskParameters, splashParameters, trialParameters)

function [experimentRecord timeStamps] = maskedPrimeTabletMain
(experimentParameters, maskParameters, splashParameters, trialParameters)

% ARGUMENTS - all structs containing various bits of information
% experimentParameters: various experiment parameters (eg subID, nBlocks)
% maskParameters:      mask information for all trials
% splashParameters:    parameters for the splashScreen
% trialParameters:     one blocks's worth of trial parameters. Used
%                       as per stimuli
%
%
% OUTPUT - optional, if you want it sent to the command window
% experimentRecord: cell array detailing the order in which the trials
%                  were presented + responses, scores and RTs
% timeStamps:       [primeStimOnset maskStimOnset blankStimOnset
%                  targetStimOnset blank2StimOnset]
%
% Contains two subfunctions:
%   cleanup
%   centreText
% Andrew D Wilson 2008

```

```
function primingMain(parameterFileName, subID)
% function primingMain(parameterFileName, subID)
%
% The primary interface between an experimenter and an experiment.
% Handles reading in the parameters from the text file
% 'parameterFileName' and calls the correct code to run the specified
% experiment.
%
% ARGUMENTS
% parameterFileName: a string specifying the full (i.e including the
%                    .txt suffix) file name of the tab-delimited
%                    parameter file.
% subID:             a string specifying the subject ID code (eg
%                    'sub01') that will be used to save data out, etc
%
% Andrew D Wilson 2008
```

```

function [experimentParameters maskParameters splashParameters trialParameters]
= readExperimentParameters(fileName)
% function [experimentParameters maskParameters trialParameters] =
%     readExperimentParameters(fileName)
%
% Reads in 'fileName', a tab delimited version of
% ExperimentParameters.xls which has been edited for the current
% experiment. The file format must not be changed - this code requires
% that everything is where the template puts it.
%
% ARGUMENTS
% fileName: a string specifying the full, exact name of the parameter
%           file (eg 'ExperimentParameters.txt')
%
% RETURN VALUES
% Three separate structs, with fields named for the headings in the
% .xls file:
% experimentParameters -
%     subjectID          = empty, to be filled by
%                         primingMain.m.
%     fileName           = stores the file name of the
%                         parameter file that generated this
%                         experiment
%     ResponseDevice      = a string specifying 'button' or
%                         'tablet'
%     Masking             = a string specifying 'masked' or
%                         'unmasked'
%     MaskTargetISI       = a string specifying 'yes' or 'no'
%                         about the mask/target ISI
%     Randomisation       = a string specifying 'yes' or 'no'
%                         about whether to randomise trial
%                         presentation
%     TabletStartLocation = a location code for the
%                         'home' location in tablet
%                         experiments
%     nBlocks             = a number specifying the number of
%                         blocks
%     nRepetitionsPerTrialType = a number specifying how many of
%                         each unique trial type each block
%                         should have.
%     nTrials eventually equals nBlocks * nRepetitionsPerTrialType *
%                         nUniqueTrials
%
% maskParameters (see the help for 'makeMask.m for more details)
%     maskType           = a string specifying the type of mask ('lines',
%                         'squares')
%     xExtentPixels      = a number specifying the size of the mask, in
%                         pixels, along the x axis
%     yExtentPixels      = a number specifying the size of the mask, in
%                         pixels, along the x axis
%     boxSize            = a number specifying how big each sub-box should
%                         be within the mask.
%     meanPenWidth       = the average width of the lines
%     penWidthSD         = the SD of the width of the lines
%     meanLineLength     = the average line length
%     lineLengthSD       = the line length SD
%
% splashParameters - details for the splash screen that displays
% between blocks
%     imperativeStimulus = a string specifying 'prime' or 'target';
%                         switches which gets demoed

```

```

% stimulusTypes      = a string specifying all the different
%                    imperative stimuli to be displayed
% orientations        = a number specifying the orientation of that
%                    stimuli
% responses           = a string specifying the response to be
%                    displayed with that stimuli
% locations           = a string specifying where to draw that
%                    example (currently requires exact
%                    coordinates, ie [x,y])
%
% trialParameters - note that the .txt file should contain one row for
% each unique trial type. The resulting struct from here reads that in
% nRepetitionsPerTrialType times
% UniqueTrialNumbers = a number identifying from which trial in
%                    fileName a given trial was created
% TrialName            = a column of strings for the trials
% TrialType            = a column of strings specifying the
%                    prime/target compatibility
% Feedback            = a column of 0s or 1s to switch feedback on
%                    or off
% Prime              = a string specifying 'arrow', 'doubleArrow',
%                    'brackets' (ie prime type)
% PrimeDuration        = a number specifying the number of frames
%                    the prime should be presented
% PrimeLocation        = a string specifying 'up', 'down', 'left',
%                    'right' or 'centre' (position relative to
%                    the mask)
% PrimeSize           = a gain on size. makeStimuli.m multiplies
%                    the default sizes by this to alter the size
% PrimeOrientation     = a number, in degrees, specifying the
%                    orientation. 0 = default way up, increasing
%                    clockwise
% MaskDuration         = a number specifying the number of frames,
%                    the mask (if used) should be presented
% MaskLocation         = a string specifying 'up', 'down', 'left',
%                    'right' or 'centre' (position relative to
%                    the screen centre)
% BlankDuration        = a number specifying the number of frames
%                    for which the blank after the mask should
%                    be presented
% Target, TargetDuration, TargetLocation, TargetSize, TargetOrientation
%                    = matching parameters for the target
% CorrectResponse      = a string specifying 'left', 'right' - used
%                    to score the trial
%
% Andrew D. Wilson, 2009

```

```
function data = sortData(experimentRecord)
% function data = sortData(experimentRecord)
%
% Takes an experimentRecord file (output from an experiment) with RTs
% in it and computes the median RT for the correct trials, sorted
% by trialType. It also counts the errors and misses and outputs them
% as a proportion of the number of trials in that condition.
%
% Designed primarily at this point to handle button press data;
% requires testing to handle scored tablet data
%
% Andrew D Wilson 2009
```

```

function splashScreen(wPtr, splashParameters, trialParameters, stimuli)
% function splashScreen(wPtr, splashParameters, trialParameters, stimuli)
%
% Draws a splash screen for use at the start of an experiment and in
% between blocks. It uses the parameters from the text file to find
% the information required to draw the right stimuli, places the
% specified correct response under the right stimulus, and puts
% everything where you tell it in 'Location'. It also draws the text
% 'Press the space bar to continue' centred and at y = 20% of the
% screen height up from the bottom)
%
% NB: Location currently requires you to specify the specific
% coordinates where you want things to go. I will eventually implement
% some defaults (left, right, top, down, centre) all defined relative
% to the centre, as well as any other requests for commonly used
% locations. This is low priority, though. Use the function
% utilities('screenSize') to find the size of your monitor in pixels,
% and use this to tweak where you want things. Remember: (0,0) is the
% top left corner, x increases going right and y increases going down.
%
% ARGUMENTS
% wPtr:           the window pointer for where the stimuli are to be
%                 drawn. This will generally be the main wPtr for an
experiment
% splashParameters: a struct containing the parameters read in from
%                 the text parameter file for the splash screen. See
%                 'help readExperimentParameters' for more information
% trialParameters:  a struct containing the parameters read in from
%                 the text parameter file for all trials. See 'help
%                 readExperimentParameters for more information
% stimuli:         a struct containing prime and target coordinates.
%                 See 'help makeStimuli' for more information
%
% Andrew D Wilson (2009)

```



```
function test(fileName)
%function test(fileName)
% Runs through one unrandomised block of the experiment specified in
% fileName with 2s intervals between stimuli. This allows you to check
% that all your stimuli are drawing correctly, and are sized and
% located where you want, plus to make sure your primes are
% displaying, etc.
%
% Escape key functionality is enabled whenever something is displayed
% on the screen (prime, mask or target)
%
% Andrew D Wilson (2009)
```

```

function [experimentRecord timeStamps] = unmaskedPrimeForceMain
(experimentParameters, splashParameters, trialParameters)

function [experimentRecord timeStamps] = unmaskedPrimeMain
(experimentParameters, splashParameters, trialParameters)

function [experimentRecord timeStamps] = unmaskedPrimeTabletMain
(experimentParameters, splashParameters, trialParameters)

% ARGUMENTS - all structs containing various bits of information
% experimentParameters: various experiment parameters (eg subID,
%                       nBlocks)
% splashParameters:    parameters for the splashScreen
% trialParameters:     one blocks's worth of trial parameters. Used
%                       as per stimuli
%
% RETURN VALUES - optional, if you want it sent to the command window
% experimentRecord: cell array detailing the order in which the trials
%                   were presented + responses, scores and RTs
% timeStamps:        [primeStimOnset maskStimOnset blankStimOnset
%                     targetStimOnset blank2StimOnset]
%
% Contains two subfunctions:
%   cleanup
%   centreText
% Andrew D Wilson 2008

```

```
function utilities(subFunction)
% function utilities(subFunction)
%
% Contains a set of minor functions that are of use in setting up and
% configuring the Masked Priming Toolbox.
%
% ARGUMENT:
% subfunction: a string specifying which utility function you wish to
% call. Currently implemented:
%   'keyName':    returns the name of the key you press via KbName;
%                 use this to find the right value for the parameter
%                 CorrectResponse
%   'screenSize': returns the size, in pixels, of the primary monitor
%
% Andrew D Wilson 2009
```

## **Change Log**

### **V1.0**

Fully functional beta version of code released. Can run keyboard, tablet and force transducer experiments with slash screens, etc Data analysis on keyboard and tablet data fully implemented, force data analysis to come.

## **Features to Add**

### **Data Analysis**

Compute path length

Force data analysis

Trial by trial kinematic + force analysis to allow scoring, feedback, etc to occur online

Tablet: output average trajectories by condition, individual trajectories labelled by trial number for sequential dependency analyses, plus RT and kinematics

### **Stimuli**

Identical masks (i.e. one time random generation)

Mobile stimuli (e.g. moving masks) – use maskLocation as an initial location and spend x number of frames moving smoothly to a final location (decided by n frames or perhaps by specifying maskLocation as [start,stop]).

Colour of everything. Currently default to black on white; code up basic colours as defaults plus the ability to specify an RGB triplet. Prime, mask, and target should all get a colour per trial – background perhaps as ExperimentParameter? Or early TrialParameter?

### **Code**

Condense masked and unmasked versions to a single file, switch behaviour on mask type there.

There is error checking information available to confirm the timing of a given trial - this should all get used to replace trials on the fly if for any reason a timing deadline is missed.

Parameterise or find some way to automate setting the COM port number for force experiments.

Default locations in splashScreen.m