Supplemental materials for:

An integrated theory of deciding and acting

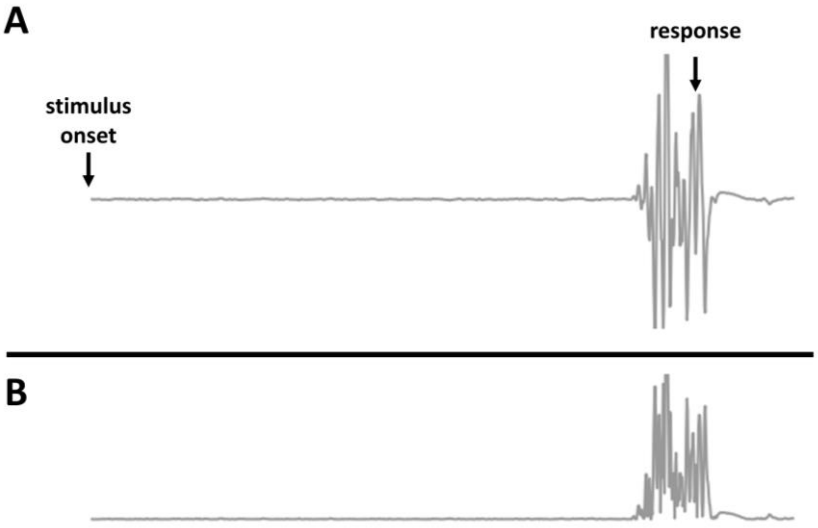
Mathieu Servant[1*], Gordon D. Logan[2], Thibault Gajdos[3], Nathan J. Evans[4]

[1]Laboratoire de Recherches Intégratives en Neurosciences et Psychologie Cognitive UR 481 et MSHE Ledoux USR 3124, Université de Franche-Comté, France

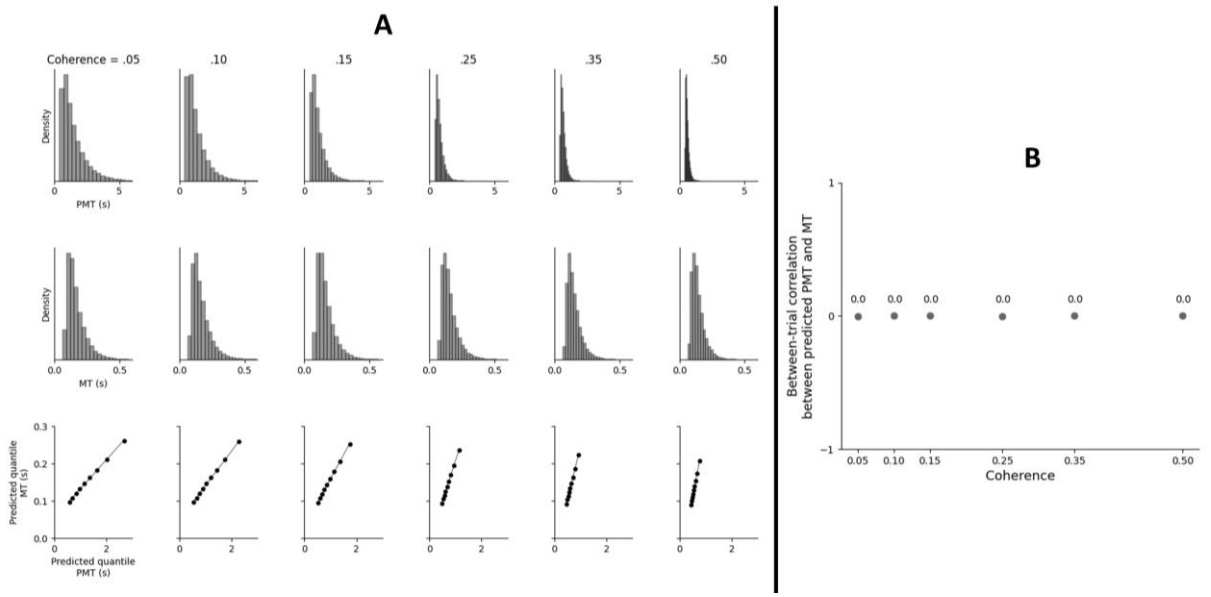
[2]Department of Psychological Sciences, Vanderbilt University, USA

[3]Laboratoire de Psychologie Cognitive UMR 7286, Aix-Marseille Université, France

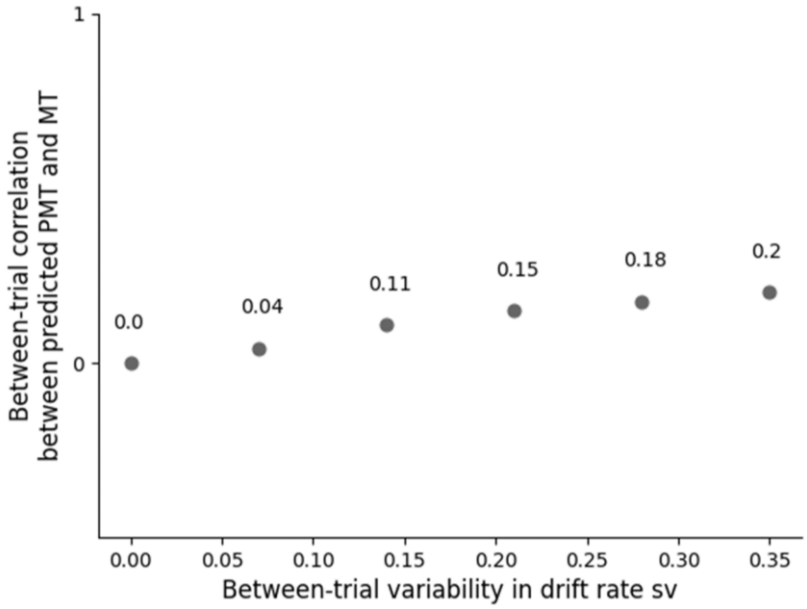
[4]School of Psychology, University of Queensland, Australia



Supplementary Figure S1. Illustration of an EMG burst in its raw form (**A**) and after rectification (**B**). Raw bursts are characterized by a discrete onset, followed by a sequence of increases and decreases in voltage that occur on the muscle fiber membrane. To analyze their electrical properties, researchers take the absolute value of voltages across time points, a procedure known as rectification. Rectified EMG amplitude scales with the level of global muscle excitation (Vigotsky et al., 2018).



Supplementary Figure S2. In order to examine the shape of predicted PMT and MT distributions and the between-trial correlation between the two chronometric variables, DTDM was simulated using best-fitting parameters averaged across subjects reported by Ratcliff & McKoon (2008, Experiment 1) in a random dot motion task featuring six levels of motions coherence (.05, .10, .15, .25, .35, .50). No between-trial variability in drift rate, starting point, and non-decision time was incorporated to focus on the main components of the model. With a diffusion coefficient set to 0.1, the parameters were: drift rate *v*5% = 0.042, *v*10% = 0.079, *v*15% = 0.133, *v*25% = 0.227, *v*35% = 0.291, *v*50% = 0.369; upper response bound *r* = 0.111 (= upper decision threshold reported by Ratcliff and McKoon). The upper EMG bound *m* was set to 50% of the upper response bound *r*. Mean nondecision time (*Ter* = 0.418 s) was decomposed into sensory encoding and corticomuscular latencies with mean *Te* = 0.368 s (added to predicted PMT) and motor latencies related to force production with mean *Tr* = 0.05 s (added to predicted MT). For each motion coherence condition, 500,000 trials were simulated using the method described in Appendix 1 and a step size *dt* = 0.001 s. **A**) Histograms of predicted PMT (row 1) and MT (row 2) in correct trials across motion coherence levels (columns). Also shown is a quantile-quantile (Q-Q) plot of predicted PMT versus MT deciles for each coherence level (row 3). Q-Q plots are approximately linear, suggesting that predicted PMT and MT distributions have a similar shape. Different values for parameter *m* do not modulate this prediction (additional simulations not shown for sake of brevity). (**B**) Between-trial Pearson's correlation coefficient between predicted PMT and MT in correct trials. The correlation is null for each motion coherence level, reflecting the Markov property of the diffusion process (see main text).



Supplementary Figure S3. Between-trial correlation between predicted PMT and MT when between-trial variability in drift rate increases (from 0 to 0.35 in steps of 0.05). With the exception of this parameter, DTDM was simulated in the same way as in Supplementary Figure S2.

Appendix 1: DTDM code

This appendix is structured in two parts. Part 1 shows a C implementation of DTDM. Part 2 shows how to call the C code from Python. The model is simulated using the method and framework of Evans (2019), a time step *dt* = 0.001 s, and a diffusion coefficient fixed at 0.1. In each trial, the predicted PMT is defined as the sum of a residual latency *Te* and the latency between accumulation onset and the last crossing of an EMG bound. The predicted MT is defined as the sum of a residual latency *Tr* and the latency between the last crossing of an EMG bound and the corresponding response bound.

Part 1: C implementation of DTDM

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>
#include <math.h>


void DTDM(double z, double v,double rU, double mU, double Te, double Tr,
          double sv, double sz, double sTe, double sTr,
          double s, double dt, double *resp, double *RT,
          double *PMT, double *MT,
          int n, int maxiter,
          int rangeLow, int rangeHigh, double *randomTable)
{
  double rhs,x,randNum,sampleV,sampleTe,sampleTr,rL,mL;
  int i,iter,blah,outOfBounds;

  // DTDM parameters
  // z: starting point of the accumulation process
  // v: drift rate
  // rU: upper response bound
  // rL: lower response bound
  // mU: upper EMG bound
  // mL: lower EMG bound
  // Te: mean duration of sensory encoding and corticomusculuar delay (Te
is added to predicted PMT)
  // Tr: mean duration of residual motor components related to force
production (Tr is added to predicted PMT)
  //sv: between-trial variability in drift rate (SD of normal distribution
with mean v)
  //sz: between-trial variability in starting point (range of uniform with
mean z)
  //sTe: between-trial variability in Te (range of uniform with mean Te)
  //sTr: between-trial variability in Tr (range of uniform with mean Tr)
  //s: diffusion coefficient

  //Other important variables:
  //dt: step size dt (in seconds)
  //n: number of simulated trials
  //outOfBounds: 0 if the decision variable is located in the region
defined by lower and upper EMG bounds. 1 if if the decision variable is
located in the region defined by an EMG bound and the corresponding
response bound
```

```
  //resp: vector containing the predicted accuracy for each trial (1 if
correct response, 2 if incorrect response, -1 if the sample path did not
hit a response bound within the allocated number of time steps (defined by
maxiter)
  //RT: vector containing the predicted RT for each trial
  //PMT vector containing the predicted PMT for each trial
  //PMT vector containing the predicted PMT for each trial
  //The variables rangeLow, rangeHigh, and randomTable serve to simulate a
random draw from a standard normal distribution (see python code and
corresponding method described in Evans (2019).


  // randomize seed of random number generator
  struct timeval t1;
  gettimeofday(&t1, NULL);
  srand(t1.tv_usec * t1.tv_sec);


  rhs=sqrt(dt)*(s);
  rL = -rU;
  mL = -mU;

  // simulate n trials
  for (i=0;i<n;i++) {

    if (sz < 0.00001) {
        x=z;
    } else {
        x = z + ((rand()/(1.0 + RAND_MAX))*sz) - (sz/2);  //rand()/(1.0 +
RAND_MAX) simulates a random number from a uniform distribution bounded at
0 and 1
    }

    if (sv < 0.00001) {
        sampleV=v;
    } else {
        randNum = rand()/(1.0 + RAND_MAX);
        blah = (rangeHigh) - (rangeLow) + 1;
        blah = (randNum * blah) + (rangeLow);
        randNum = randomTable[blah];//randNum is a random number from the
standard normal distribution
        sampleV = v + (sv*randNum);
    }

    if (sTe < 0.00001) {
        sampleTe=Te;
    } else {
        sampleTe = Te + ((rand()/(1.0 + RAND_MAX))*sTe) - (sTe/2);
    }
    if (sTr < 0.00001) {
        sampleTr=Tr;
    } else {
        sampleTr = Tr + ((rand()/(1.0 + RAND_MAX))*sTr) - (sTr/2);
    }

    iter=0;
    resp[i]=(double) -1.0;
    RT[i] = (double) -1.0;
    PMT[i] = (double) -1.0;
    MT[i] = (double) -1.0;
    outOfBounds=0;
```

```
    do
    {
      iter = iter+1;

      randNum = rand()/(1.0 + RAND_MAX);
      blah = (rangeHigh) - (rangeLow) + 1;
      blah = (randNum * blah) + (rangeLow);
      randNum = randomTable[blah];

      x = x + (sampleV*dt) + (rhs*randNum);//decision variable

      if ((x<=mL || x>=mU) && (outOfBounds==0)){
        outOfBounds=1;// the decision variable is now located in the region
between an EMG bound and the corresponding response bound
        PMT[i] = (((double) iter)*dt - dt/((double) 2.0)) + sampleTe;//in
case several EMG bound hits occur, PMT will store the latency of the last
hit before the response
      }
      if (outOfBounds==1) {
        if (x<mU && x>mL) {//check if the decision variable has left the
region between EMG and response bounds
          outOfBounds=0;//if this is the case, reset the outOfBounds
variable to 0
        }
      }
      if (x>=rU) {
        resp[i]=(double) 1.0 ; //correct response
        break;
      }
      if (x<=rL) {
        resp[i]=(double) 2.0 ; //incorrect response
        break;
      }
    } while (iter<maxiter) ;//simulate trial until max number of dt steps
allowed (defined by maxiter)

    RT[i]=(((double) iter)*dt - dt/((double) 2.0)) + sampleTe + sampleTr;
    MT[i] = RT[i]-PMT[i];
  }
}
```

Part 2: Call of DTDM C code from Python

We first need to compile the C code into a dynamic library (on a Linux operating system). This can be done using the following command:

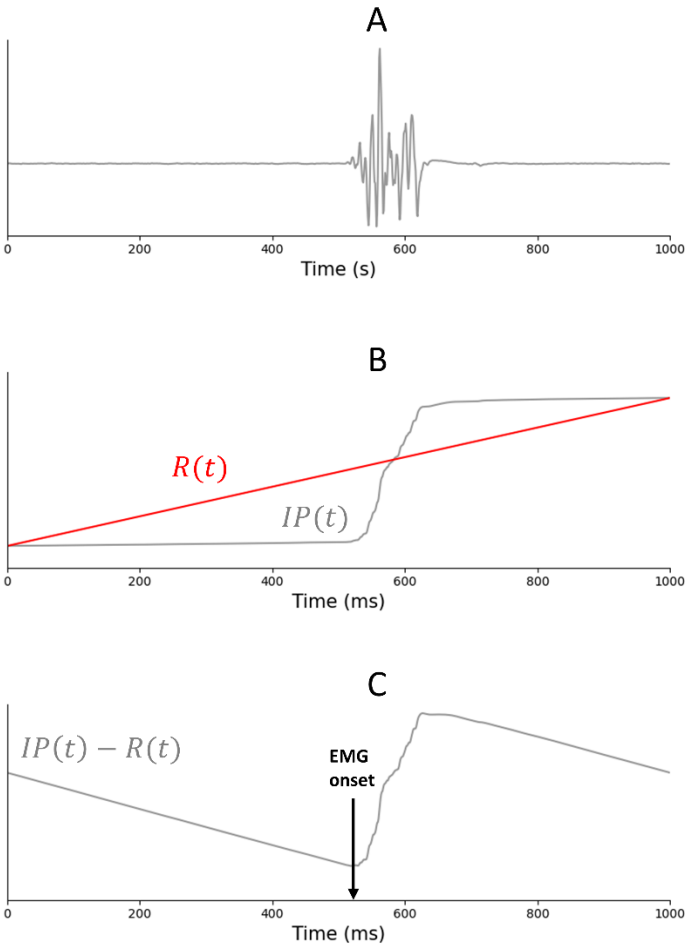
```
gcc -shared -o DTDM.so -fPIC DTDM.c
```

where DTDM.c is the name of the file containing our C code. Below is a Python code to call the model, using 100,000 simulated trials and a time step *dt* = 0.001 s:

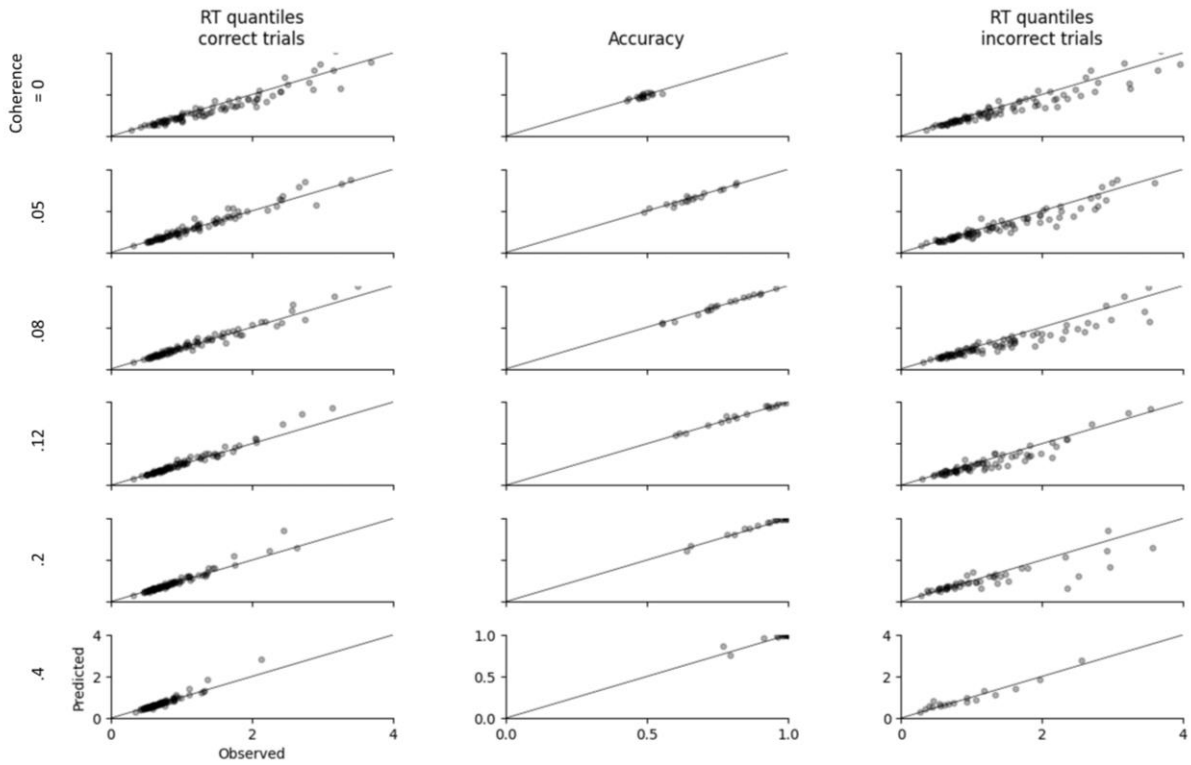
```
import ctypes
import numpy as np
from scipy.stats import norm
from copy import deepcopy
import os
np.set_printoptions(5, suppress=True)
```

```
#load DTDM using Python's ctypes library
myso = ctypes.cdll.LoadLibrary(r"/Work/Users/mservant/code_DTDM.so")
DTDM = myso.DTDM

#define a convenient class to store model parameters
class params:
        def __init__(self, v, rU, mU, Te, Tr, sv=0, sz=0,sTe=0, sTr=0):
            self.v = v
            self.z = 0#unbiased starting point of the accumulation process
            self.rU = rU
            self.mU = mU
            self.Te = Te
            self.Tr = Tr
            self.sv = sv
            self.sz = sz
            self.sTe = sTe
            self.sTr = sTr
            self.s = .1
            self.dt = .001
            self.n = 100000
            self.resp = np.zeros(self.n)
            self.RT = np.zeros(self.n)
            self.PMT = np.zeros(self.n)
            self.MT = np.zeros(self.n)
            self.maxiter = 15000
            #LUT parameters (see Evans, 2019)
            #build LUT table for gaussian random number generator
            interval = .0001
            gran = np.arange(interval, 1, interval)
            use_table = norm.ppf(gran)
            self.randomTable = use_table
            self.rangeLow = 0
            self.rangeHigh = len(self.randomTable)

#define model parameters
v = 0.369
rU = 0.111
mU = .5*0.111
Te = 0.418-.05
Tr = .05

#instantiate params class
obj = params(v, rU, mU, Te, Tr)
#call model
DTDM(
    ctypes.c_double(obj.z),
    ctypes.c_double(obj.v),
    ctypes.c_double(obj.rU),
    ctypes.c_double(obj.mU),
    ctypes.c_double(obj.Te),
    ctypes.c_double(obj.Tr),
    ctypes.c_double(obj.sv),
    ctypes.c_double(obj.sz),
    ctypes.c_double(obj.sTe),
    ctypes.c_double(obj.sTr),
    ctypes.c_double(obj.s),
    ctypes.c_double(obj.dt),
    ctypes.c_void_p(obj.resp.ctypes.data),
    ctypes.c_void_p(obj.RT.ctypes.data),
    ctypes.c_void_p(obj.PMT.ctypes.data),
    ctypes.c_void_p(obj.MT.ctypes.data),
```

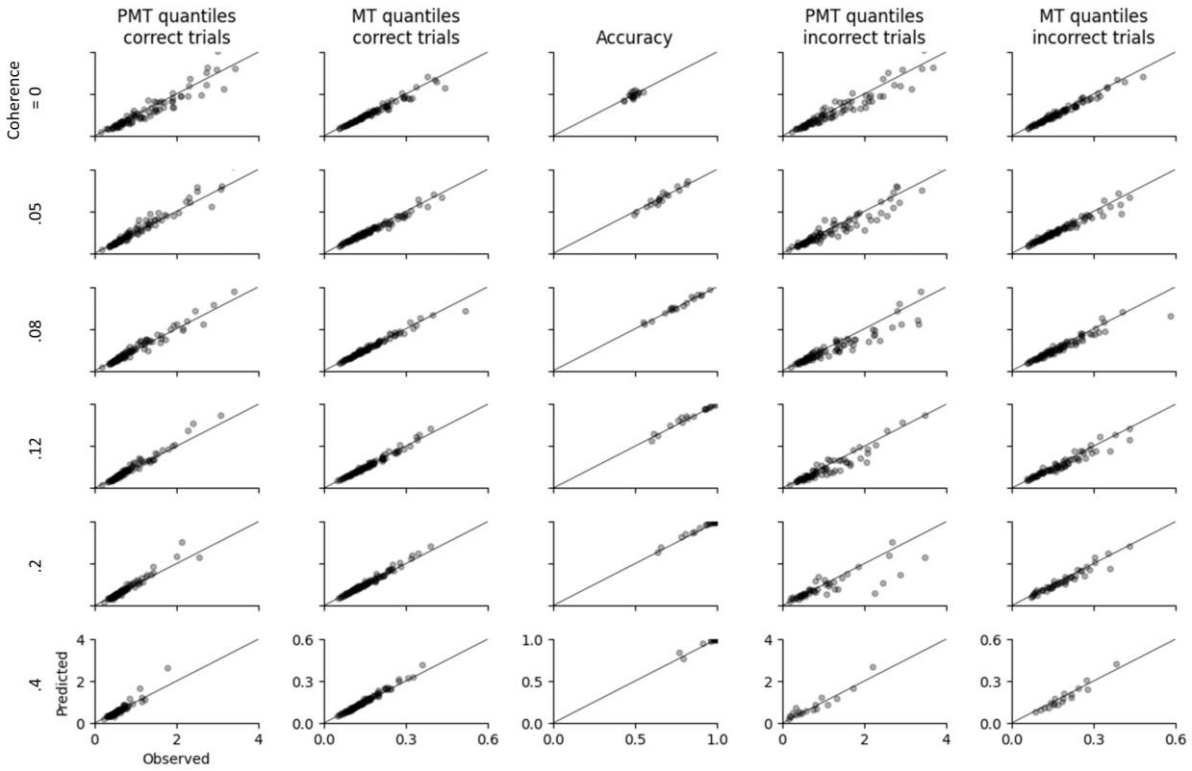
```
        ctypes.c_int(obj.n),
        ctypes.c_int(obj.maxiter),
        ctypes.c_int(obj.rangeLow),
        ctypes.c_int(obj.rangeHigh),
        ctypes.c_void_p(obj.randomTable.ctypes.data))

    #perform a deep copy of results
    pred_MT = deepcopy(obj.MT)
    pred_resp = deepcopy(obj.resp)
    pred_PMT = deepcopy(obj.PMT)

    #only keep trials that have converged
    pred_MT = pred_MT[pred_resp > 0]
    pred_PMT = pred_PMT[pred_resp > 0]
    pred_resp = pred_resp[pred_resp > 0]

    #print mean PMT in correct trials:
    print(np.mean(pred_PMT[pred_resp==1]))
    #print mean MT in correct trials:
    print(np.mean(pred_MT[pred_resp==1]))
```



Supplementary Figure S4. Illustration of the integrated profile method to detect the onset of an EMG burst shown in **A**. The cumulative sum of rectified voltages $IP(t)$ is first computed, along with the reference line $R(t)$ (**B**; see main text for equations of $IP(t)$ and $R(t)$). The EMG onset corresponds to the minimum of the difference $IP(t) - R(t)$ shown in **C**.



Supplementary Figure S5. Individual DM fits to the RT distributions of correct and error responses and to accuracy data in each coherence condition. Values along the identity line indicate correspondence between observed (x-axis) and predicted data (y-axis).



Supplementary Figure S6. Individual DTDM fits to the joint distributions of PMT and MT in correct and error trials and to accuracy data in each coherence condition. Values along the identity line indicate correspondence between observed (x-axis) and predicted data (y-axis).

References

Evans, N. J. (2019). A method, framework, and tutorial for efficiently simulating models of decision-making. *Behavior Research Methods*, *51*(5), 2390–2404. https://doi.org/10.3758/s13428-019-01219-z

Ratcliff, R., & McKoon, G. (2008). The Diffusion Decision Model: Theory and Data for Two-Choice Decision Tasks. *Neural Computation*, *20*(4), 873–922. https://doi.org/10.1162/neco.2008.12-06-420

Vigotsky, A. D., Halperin, I., Lehman, G. J., Trajano, G. S., & Vieira, T. M. (2018). Interpreting Signal Amplitudes in Surface Electromyography Studies in Sport and Rehabilitation Sciences. *Frontiers in Physiology*, *8*. https://doi.org/10.3389/fphys.2017.00985