Probabilistic Programming: From Principled Foundations, Through Efficient Implementation, To Innovative Applications

Jeffrey Mark Siskind

Purdue University

Thursday 10 July 2014

This research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-10-2-0060. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either express or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation herein.

Outline

Outline

Probabilistic Scheme

PROBABILISTIC SCHEME ≡ SCHEME

Sussman, G.J. and Steele, Jr., G.L., *Scheme: an interpreter for extended lambda calculus*, AI Memo 349, MIT, 1975.

$$\textsc{Probabilistic Scheme} \equiv \textsc{Scheme} + \boxed{\begin{array}{c}\text{Bernoulli}\\ \text{Trials}\end{array}}$$

Sussman, G.J. and Steele, Jr., G.L., *Scheme: an interpreter for extended lambda calculus*, AI Memo 349, MIT, 1975.

$$\text{PROBABILISTIC SCHEME} \equiv \text{SCHEME} + \boxed{\begin{array}{c}\text{Bernoulli}\\ \text{Trials}\end{array}} + \boxed{\begin{array}{c}\text{Partition}\\ \text{Function}\end{array}}$$

Sussman, G.J. and Steele, Jr., G.L., *Scheme: an interpreter for extended lambda calculus*, AI Memo 349, MIT, 1975.

Radul, A., *Report on the probabilistic language Scheme*, DLS, 2007.

```
(define (fold-distribution-thunk f i thunk)
 (call-with-current-continuation
  (lambda (c)
   (let ((accumulation i)
         (p 1)
         (saved-flip *flip*)
         (saved-bottom *bottom*)
         (saved-current-probability *current-probability*))
    (set! *flip*
          (lambda (alpha)
           (unless (<= 0 alpha 1) (error #f "Alpha not probability"))
           (cond ((zero? alpha) #f)
                 ((= alpha 1) #t)
                 (else
                  (call-with-current-continuation
                   (lambda (c)
                    (let ((saved-p p) (saved-bottom *bottom*))
                     (set! p (* alpha p))
                     (set! *bottom*
                           (lambda ()
                            (set! p (* (- 1 alpha) saved-p))
                            (set! *bottom* saved-bottom)
                            (c #f)))
                     #t)))))))
    (set! *bottom*
          (lambda ()
           (set! *flip* saved-flip)
           (set! *bottom* saved-bottom)
           (set! *current-probability* saved-current-probability)
           (c accumulation)))
    (set! *current-probability* (lambda () p))
    (let ((value (thunk))) (set! accumulation (f value p accumulation)))
    (bottom)))))

(define-syntax fold-distribution
 (syntax-rules ()
  ((fold-distribution f i e) (fold-distribution-thunk f i (lambda () e)))))
```

```
(define-syntax distribution ...)

(define-syntax support ...)

(define-syntax probability
 (syntax-rules ()
  ((probability e)
   (fold-distribution
    (lambda (value p accumulation) (if value (+ p accumulation) accumulation))
    0
    e))))

(define-syntax expected-value ...)

(define-syntax entropy ...)

(define-syntax most-likely-value-in-bag ...)

(define-syntax probability-of-most-likely-value-in-bag
 (syntax-rules ()
  ((probability-of-most-likely-value-in-bag e)
   (fold-distribution
    (lambda (value p accumulation) (max p accumulation)) 0 e))))
```

```
(define-syntax distribution ...)

(define-syntax support ...)

(define-syntax probability
 (syntax-rules ()
  ((probability e)
   (fold-distribution
    (lambda (value p accumulation) (if value (+ p accumulation) accumulation))
    0
    e))))

(define-syntax expected-value ...)

(define-syntax entropy ...)

(define-syntax most-likely-value-in-bag ...)

(define-syntax probability-of-most-likely-value-in-bag
 (syntax-rules ()
  ((probability-of-most-likely-value-in-bag e)
   (fold-distribution
    (lambda (value p accumulation) (max p accumulation)) 0 e))))
```

217 lines in R6RS

Discrete Graphical Model Scheme

Discrete Graphical Model Scheme $\equiv$ Probabilistic Scheme

$$\text{DISCRETE GRAPHICAL MODEL SCHEME} \equiv$$
$$\text{PROBABILISTIC SCHEME} + \boxed{\begin{array}{c}\text{Arc}\\ \text{Consistency}\end{array}}$$

Mackworth, A.K., *Consistency in networks of relations*, AIJ, 8(1):99–118, 1977.

$$\text{DISCRETE GRAPHICAL MODEL SCHEME} \equiv$$
$$\text{PROBABILISTIC SCHEME} + \boxed{\begin{array}{c}\text{Arc}\\ \text{Consistency}\end{array}} + \boxed{\begin{array}{c}\text{Branch and}\\ \text{Bound}\end{array}}$$

Mackworth, A.K., *Consistency in networks of relations*, AIJ, 8(1):99–118, 1977.

Land, A.H. and Doig, A.G., *An automatic method of solving discrete programming problems*, Econometrica, 28(3):497–520, 1960.

```
(define-record-type distribution-variable
  (fields (mutable distribution) (mutable demons)))

(define (assert-constraint-ac! constraint ds)
 (for-each
  (lambda (d)
   (attach-demon!
    (lambda ()
     (for-each-indexed
      (lambda (d i)
       (restrict-distribution!
        d
        (the-elements
         (lambda (x)
          (let loop ((ds ds) (xs '()) (j 0))
           (if (null? ds)
               (apply constraint (reverse xs))
               (if (= j i)
                   (loop (rest ds) (cons x xs) (+ j 1))
                   (some-element
                    (lambda (x) (loop (rest ds) (cons x xs) (+ j 1)))
                    (first ds))))))
         d)))
      ds))
    d))
  ds))

(define (stochastic-solution ds)
 (let loop ((ds ds) (xs '()))
  (if (null? ds)
      (reverse xs)
      (let ((pair
             (draw-pair (distribution-variable-distribution (first ds)))))
       (restrict-distribution! (first ds) (list pair))
       (loop (rest ds) (cons (first pair) xs))))))
```

```
(define (stochastic-branch-and-bound-solution ds)
 (let ((best 0))
  (let loop ((ds ds) (xs '()))
   (when (<= (fold (lambda (v d)
                     (* v
                        (map-reduce
                         max 0 cdr (distribution-variable-distribution d))))
                   (current-probability)
                   ds)
             best)
    (bottom))
   (cond ((null? ds)
          (set! best (current-probability))
          (reverse xs))
         (else
          (let ((pair
                 (draw-pair (distribution-variable-distribution (first ds)))))
           (restrict-distribution! (first ds) (list pair))
           (loop (rest ds) (cons (first pair) xs)))))))))
```

```
(define (stochastic-branch-and-bound-solution ds)
 (let ((best 0))
  (let loop ((ds ds) (xs '()))
   (when (<= (fold (lambda (v d)
                     (* v
                        (map-reduce
                         max 0 cdr (distribution-variable-distribution d))))
                   (current-probability)
                   ds)
             best)
    (bottom))
   (cond ((null? ds)
          (set! best (current-probability))
          (reverse xs))
         (else
          (let ((pair
                 (draw-pair (distribution-variable-distribution (first ds)))))
           (restrict-distribution! (first ds) (list pair))
           (loop (rest ds) (cons (first pair) xs))))))))
```

206 lines in R6RS

SCHWISH

SCHWISH ≡ DISCRETE GRAPHICAL MODEL SCHEME

SCHWISH $\equiv$ DISCRETE GRAPHICAL MODEL SCHEME + WISH

Ermon, S., Gomes, C., Sabharwal, A., and Selman, B., *Taming the curse of dimensionality: discrete integration by hashing and optimization*, ICML, 2013

```
(define (assert-random-xor-constraint! ds)
 (let ((b (random-boolean)))
  (apply assert-stochastic-constraint!
         (lambda vs (eq? (fold xor #f vs) b))
         (random-subset ds))))

(define (wish-find-constrained-max ds i)
 (most-likely-probability
  (begin (for-each-n (lambda (j) (assert-random-xor-constraint! ds)) i)
         (stochastic-branch-and-bound-solution ds))))

(define (wish ds delta alpha)
 (let* ((m (length ds))
        (xs (map-n
             (lambda (i)
              (median
               (map-n (lambda (t) (wish-find-constrained-max ds i))
                      (inexact->exact (ceiling (/ (log (/ m delta)) alpha))))))
             (+ m 1))))
  (+ (first xs) (summation (lambda (x i) (* x (expt 2.0 i))) (rest xs)))))
```

```
(define (assert-random-xor-constraint! ds)
 (let ((b (random-boolean)))
  (apply assert-stochastic-constraint!
         (lambda vs (eq? (fold xor #f vs) b))
         (random-subset ds))))

(define (wish-find-constrained-max ds i)
 (most-likely-probability
  (begin (for-each-n (lambda (j) (assert-random-xor-constraint! ds)) i)
         (stochastic-branch-and-bound-solution ds))))

(define (wish ds delta alpha)
 (let* ((m (length ds))
        (xs (map-n
             (lambda (i)
              (median
               (map-n (lambda (t) (wish-find-constrained-max ds i))
                      (inexact->exact (ceiling (/ (log (/ m delta)) alpha))))))
             (+ m 1))))
  (+ (first xs) (summation (lambda (x i) (* x (expt 2.0 i))) (rest xs)))))
```

49 lines in R6RS

VLAD

VLAD ≡ SCHEME

VLAD $\equiv$ SCHEME $+ \nabla$

Wengert, R.E., *A simple automatic derivative evaluation program*, CACM, 7(8):463–4, 1964.

Speelpenning, B., *Compiling fast partial derivatives of functions given by algorithms*, UIUC, 1980.

Sussman, G.J, Wisdom, J., and Mayer, M.E., *Structure and interpretation of classical mechanics*, MIT Press, 2001.

$$\text{VLAD} \equiv \text{SCHEME} + \nabla + \boxed{\begin{array}{c}\text{Gradient based}\\ \text{Optimization}\end{array}}$$

Wengert, R.E., *A simple automatic derivative evaluation program*, CACM, 7(8):463–4, 1964.

Speelpenning, B., *Compiling fast partial derivatives of functions given by algorithms*, UIUC, 1980.

Sussman, G.J, Wisdom, J., and Mayer, M.E., *Structure and interpretation of classical mechanics*, MIT Press, 2001.

Wolfe, P., *The reduced gradient method*, RAND, 1962.

Wolfe, P., *Methods of nonlinear programming*, in Abadie, J., ed., 'Nonlinear programming,' pp. 97–131, Wiley, 1967.

$$\text{VLAD} \equiv \text{SCHEME} + \nabla + \boxed{\begin{array}{c}\text{Gradient based}\\ \text{Optimization}\end{array}}$$

$$\arg\max_{\theta} \Pr(x|\theta)$$

Wengert, R.E., *A simple automatic derivative evaluation program*, CACM, 7(8):463–4, 1964.

Speelpenning, B., *Compiling fast partial derivatives of functions given by algorithms*, UIUC, 1980.

Sussman, G.J, Wisdom, J., and Mayer, M.E., *Structure and interpretation of classical mechanics*, MIT Press, 2001.

Wolfe, P., *The reduced gradient method*, RAND, 1962.

Wolfe, P., *Methods of nonlinear programming*, in Abadie, J., ed., 'Nonlinear programming,' pp. 97–131, Wiley, 1967.

$$\text{VLAD} \equiv \text{SCHEME} + \nabla + \boxed{\begin{array}{c}\text{Gradient based}\\ \text{Optimization}\end{array}}$$

$$\arg\max_{\theta} \Pr(x|\theta)$$

Wengert, R.E., *A simple automatic derivative evaluation program*, CACM, 7(8):463–4, 1964.

Speelpenning, B., *Compiling fast partial derivatives of functions given by algorithms*, UIUC, 1980.

Sussman, G.J, Wisdom, J., and Mayer, M.E., *Structure and interpretation of classical mechanics*, MIT Press, 2001.

Wolfe, P., *The reduced gradient method*, RAND, 1962.

Wolfe, P., *Methods of nonlinear programming*, in Abadie, J., ed., 'Nonlinear programming,' pp. 97–131, Wiley, 1967.

641 lines in R6RS

Outline

STALIN∇

$\textsc{Stalin}\nabla \equiv \textsc{Stalin}$

Siskind, J.M., *Flow-directed lightweight closure conversion*, TR 99-105, NEC Research Institute, 1999.

$$\text{STALIN}\nabla \equiv \text{STALIN} + \boxed{\begin{array}{c}\text{Transformation based}\\ \text{AD}\end{array}}$$

Siskind, J.M., *Flow-directed lightweight closure conversion*, TR 99-105, NEC Research Institute, 1999.

Pearlmutter, B.A. and Siskind, J.M., *Reverse-mode AD in a functional framework: lambda the ultimate backpropagator*, TOPLAS, 30(2):1–36, 2008.

$$\text{STALIN}\nabla \equiv \text{STALIN} + \boxed{\begin{array}{c}\text{Transformation based}\\ \text{AD}\end{array}} + \text{CF}(\infty)$$

Siskind, J.M., *Flow-directed lightweight closure conversion*, TR 99-105, NEC Research Institute, 1999.

Pearlmutter, B.A. and Siskind, J.M., *Reverse-mode AD in a functional framework: lambda the ultimate backpropagator*, TOPLAS, 30(2):1–36, 2008.

Shivers, III, O.G., *Control-flow analysis of higher-order languages or taming lambda*, CMU, 1991.

$$\text{STALIN}\nabla \equiv \text{STALIN} + \boxed{\begin{array}{c}\text{Transformation based}\\ \text{AD}\end{array}} + \text{CF}(\infty)$$

Siskind, J.M., *Flow-directed lightweight closure conversion*, TR 99-105, NEC Research Institute, 1999.

Pearlmutter, B.A. and Siskind, J.M., *Reverse-mode AD in a functional framework: lambda the ultimate backpropagator*, TOPLAS, 30(2):1–36, 2008.

Shivers, III, O.G., *Control-flow analysis of higher-order languages or taming lambda*, CMU, 1991.

26,384 lines in R4RS

- implement interpreter for PROBABILISTIC SCHEME in VLAD

- implement interpreter for PROBABILISTIC SCHEME in VLAD
- parameter estimation via gradient-based optimization of likelihood

- implement interpreter for PROBABILISTIC SCHEME in VLAD
- parameter estimation via gradient-based optimization of likelihood
- take gradient through interpreter

- implement interpreter for PROBABILISTIC SCHEME in VLAD
- parameter estimation via gradient-based optimization of likelihood
- take gradient through interpreter
- compile with STALIN∇

$P =$ **if** x_0 **then** 0 **else if** x_1 **then** 1 **else** 2

$P = \textbf{if } x_0 \textbf{ then } 0 \textbf{ else if } x_1 \textbf{ then } 1 \textbf{ else } 2$

$\Pr(x_0 \mapsto \textbf{true}) = p_0$ $\quad$ $\Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$

$\Pr(x_1 \mapsto \textbf{true}) = p_1$ $\quad$ $\Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$

$P =$ **if** x_0 **then** 0 **else if** x_1 **then** 1 **else** 2

$$\Pr(x_0 \mapsto \textbf{true}) = p_0 \qquad \Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$$
$$\Pr(x_1 \mapsto \textbf{true}) = p_1 \qquad \Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$$

$$\Pr(\mathcal{E}(P) = 0 | p_0, p_1) = p_0$$
$$\Pr(\mathcal{E}(P) = 1 | p_0, p_1) = (1 - p_0)p_1$$
$$\Pr(\mathcal{E}(P) = 2 | p_0, p_1) = (1 - p_0)(1 - p_1)$$

$P =$ **if** x_0 **then** 0 **else if** x_1 **then** 1 **else** 2

$$\Pr(x_0 \mapsto \textbf{true}) = p_0 \qquad \Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$$
$$\Pr(x_1 \mapsto \textbf{true}) = p_1 \qquad \Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$$

$$\Pr(\mathcal{E}(P) = 0 | p_0, p_1) = p_0$$
$$\Pr(\mathcal{E}(P) = 1 | p_0, p_1) = (1 - p_0)p_1$$
$$\Pr(\mathcal{E}(P) = 2 | p_0, p_1) = (1 - p_0)(1 - p_1)$$

$$\prod_{v \in \{0,1,2,2\}} \Pr(\mathcal{E}(P) = v | p_0, p_1) = p_0(1 - p_0)^3 p_1(1 - p_1)^2$$

$P = \textbf{if } x_0 \textbf{ then } 0 \textbf{ else if } x_1 \textbf{ then } 1 \textbf{ else } 2$

$$\Pr(x_0 \mapsto \textbf{true}) = p_0 \qquad \Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$$
$$\Pr(x_1 \mapsto \textbf{true}) = p_1 \qquad \Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$$

$$\Pr(\mathcal{E}(P) = 0 | p_0, p_1) = p_0$$
$$\Pr(\mathcal{E}(P) = 1 | p_0, p_1) = (1 - p_0)p_1$$
$$\Pr(\mathcal{E}(P) = 2 | p_0, p_1) = (1 - p_0)(1 - p_1)$$

$$\prod_{v \in \{0,1,2,2\}} \Pr(\mathcal{E}(P) = v | p_0, p_1) = p_0(1 - p_0)^3 p_1(1 - p_1)^2$$

$$\arg\max_{p_0, p_1} \prod_{v \in \{0,1,2,2\}} \Pr(\mathcal{E}(P) = v | p_0, p_1) = \left\langle \frac{1}{4}, \frac{1}{3} \right\rangle$$

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if x0 then 0 else if x1 then 1 else 2
                   (list Pr(x0 ↦ true) = p0   Pr(x0 ↦ false) = 1 − p0
                         Pr(x1 ↦ true) = p1   Pr(x1 ↦ false) = 1 − p1
                         ...))))
   (map-reduce
    *
    1.0
    (lambda (value)
     (likelihood value tagged-distribution))
    '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if x0 then 0 else if x1 then 1 else 2
                   (list Pr(x0 ↦ true) = p0   Pr(x0 ↦ false) = 1 − p0
                         Pr(x1 ↦ true) = p1   Pr(x1 ↦ false) = 1 − p1
                         ...))))
    (map-reduce
     *
     1.0
     (lambda (value)
      (likelihood value tagged-distribution))
     '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if x0 then 0 else if x1 then 1 else 2
                   (list Pr(x0 ↦ true) = p0   Pr(x0 ↦ false) = 1 − p0
                         Pr(x1 ↦ true) = p1   Pr(x1 ↦ false) = 1 − p1
                         ...))))
   (map-reduce
    *
    1.0
    (lambda (value)
     (likelihood value tagged-distribution))
    '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if x0 then 0 else if x1 then 1 else 2
                   (list Pr(x0 ↦ true) = p0   Pr(x0 ↦ false) = 1 − p0
                         Pr(x1 ↦ true) = p1   Pr(x1 ↦ false) = 1 − p1
                         ...))))
    (map-reduce
     *
     1.0
     (lambda (value)
      (likelihood value tagged-distribution))
     '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if x0 then 0 else if x1 then 1 else 2
                   (list Pr(x0 ↦ true) = p0   Pr(x0 ↦ false) = 1 − p0
                         Pr(x1 ↦ true) = p1   Pr(x1 ↦ false) = 1 − p1
                         ...))))
    (map-reduce
     *
     1.0
     (lambda (value)
      (likelihood value tagged-distribution))
     '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if x0 then 0 else if x1 then 1 else 2
                   (list Pr(x0 ↦ true) = p0   Pr(x0 ↦ false) = 1 − p0
                         Pr(x1 ↦ true) = p1   Pr(x1 ↦ false) = 1 − p1
                         ...))))
    (map-reduce
     *
     1.0
     (lambda (value)
      (likelihood value tagged-distribution))
     '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if x0 then 0 else if x1 then 1 else 2
                   (list Pr(x0 ↦ true) = p0   Pr(x0 ↦ false) = 1 − p0
                         Pr(x1 ↦ true) = p1   Pr(x1 ↦ false) = 1 − p1
                         ...))))
    (map-reduce
     *
     1.0
     (lambda (value)
      (likelihood value tagged-distribution))
     '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if x0 then 0 else if x1 then 1 else 2
                   (list Pr(x0 ↦ true) = p0   Pr(x0 ↦ false) = 1 − p0
                         Pr(x1 ↦ true) = p1   Pr(x1 ↦ false) = 1 − p1
                         ...))))
    (map-reduce
     *
     1.0
     (lambda (value)
      (likelihood value tagged-distribution))
     '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if x0 then 0 else if x1 then 1 else 2
                   (list Pr(x0 ↦ true) = p0   Pr(x0 ↦ false) = 1 − p0
                         Pr(x1 ↦ true) = p1   Pr(x1 ↦ false) = 1 − p1
                         ...))))
    (map-reduce
     *
     1.0
     (lambda (value)
      (likelihood value tagged-distribution))
     '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if x0 then 0 else if x1 then 1 else 2
                   (list Pr(x0 ↦ true) = p0   Pr(x0 ↦ false) = 1 − p0
                         Pr(x1 ↦ true) = p1   Pr(x1 ↦ false) = 1 − p1
                         ...))))
    (map-reduce
     *
     1.0
     (lambda (value)
      (likelihood value tagged-distribution))
     '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

```
static void f2679(double a_f2679_0,double a_f2679_1,double a_f2679_2,double a_f2679_3){
  int t272381=((a_f2679_2==0.)?0:1);
  double t272406;
  double t272405;
  double t272404;
  double t272403;
  double t272402;
  if((t272381==0)){
    double t272480=(1.-a_f2679_0);
    double t272572=(1.-a_f2679_1);
    double t273043=(a_f2679_0+0.);
    double t274185=(t272480*a_f2679_1);
    double t274426=(t274185+0.);
    double t275653=(t272480*t272572);
    double t275894=(t275653+0.);
    double t277121=(t272480*t272572);
    double t277362=(t277121+0.);
    double t277431=(t277362*1.);
    double t277436=(t275894*t277431);
    double t277441=(t274426*t277436);
    double t277446=(t273043*t277441);
    ...
    double t1777107=(t1774696+t1715394);
    double t1777194=(0.-t1745420);
    double t1778533=(t1777194+t1419700);
    t272406=a_f2679_0;
    t272405=a_f2679_1;
    t272404=t277446;
    t272403=t1778533;
    t272402=t1777107;}
  else {...}
  r_f2679_0=t272406;
  r_f2679_1=t272405;
  r_f2679_2=t272404;
  r_f2679_3=t272403;
  r_f2679_4=t272402;}
```

		probabilistic-Scheme		probabilistic-Prolog	
		F	R	F	R
VLAD	STALINGRAD	1.00	1.00	1.00	1.00
ML	MLTON	106.45	124.95	789.41	483.47
	OCAML	215.73	538.68	1207.13	1534.61
	SML/NJ	197.75	272.45	2448.02	1471.94
HASKELL	GHC	■	■	■	■
SCHEME	BIGLOO	832.92	1048.11	14422.16	8286.06
	CHICKEN	2305.98	3283.00	66948.70	37792.84
	GAMBIT	879.88	1153.86	24316.03	13649.81
	IKARUS	437.46	531.10	8242.92	4845.86
	LARCENY	1651.01	1673.22	25589.62	14833.53
	MIT SCHEME	3491.10	4130.19	85819.57	48335.38
	MZC	5289.17	5929.14	154206.95	83480.27
	MZSCHEME	6235.78	7134.71	166129.12	91630.70
	SCHEME->C	682.15	794.31	10530.66	5980.27
	SCMUTILS	6456.99	■	80100.23	■
	STALIN	1240.73	1137.41	22511.79	10986.43

- ■ not implemented but could implement, including FORTRAN, C, and C++
- ■ not implemented in existing tool
- ■ can't implement

Outline

Sentence Tracker

Sentence Tracker

B: a video, represented as a set of detections in each frame, each detection annotated with features

Sentence Tracker

B: a video, represented as a set of detections in each frame, each detection annotated with features

s: a sentence, represented as a sequence of words

Sentence Tracker

B: a video, represented as a set of detections in each frame, each detection annotated with features
s: a sentence, represented as a sequence of words
Λ: a lexicon, models for each word

Sentence Tracker

B: a video, represented as a set of detections in each frame, each detection annotated with features
s: a sentence, represented as a sequence of words
Λ: a lexicon, models for each word

$$\mathcal{S}(\mathbf{B}, \mathbf{s}, \Lambda)$$

Sentence Tracker

B: a video, represented as a set of detections in each frame, each detection annotated with features
s: a sentence, represented as a sequence of words
Λ: a lexicon, models for each word

$$\mathcal{S}(\mathbf{B}, \mathbf{s}, \Lambda) = \mathbb{E}_{\Pr(\mathbf{J}|\mathbf{B})}[\Pr(\mathbf{B_J}|\mathbf{s}, \Lambda)]$$

Sentence Tracker

B: a video, represented as a set of detections in each frame, each detection annotated with features
s: a sentence, represented as a sequence of words
Λ: a lexicon, models for each word

$$\mathcal{S}(\mathbf{B}, \mathbf{s}, \Lambda) = \mathbb{E}_{\Pr(\mathbf{J}|\mathbf{B})}[\Pr(\mathbf{B_J}|\mathbf{s}, \Lambda)] = \sum_{\mathbf{J}} \Pr(\mathbf{J}|\mathbf{B}) \Pr(\mathbf{B_J}|\mathbf{s}, \Lambda)$$

Using the Sentence Tracker for Generation

Given a video clip $\mathbf{B}$ and a lexicon Λ, produce a sentential description $\mathbf{s}^*$ of the video clip.

Using the Sentence Tracker for Generation

Given a video clip $\mathbf{B}$ and a lexicon Λ, produce a sentential description $\mathbf{s}^*$ of the video clip.

$$\mathbf{s}^* = \arg\max_{\mathbf{s}} \mathcal{S}(\mathbf{B}, \mathbf{s}, \Lambda)$$

Using the Sentence Tracker for Generation

Given a video clip $\mathbf{B}$ and a lexicon Λ, produce a sentential description $\mathbf{s}^*$ of the video clip.

$$\mathbf{s}^* = \arg\max_{\mathbf{s}} \mathcal{S}(\mathbf{B}, \mathbf{s}, \Lambda)$$

Siddharth, N., Barbu, A., and Siskind, J.M., *Seeing what you're told: sentence-guided activity recognition in video*, CVPR, 2014.

Using the Sentence Tracker for Retrieval

Given a set $\{\mathbf{B}_1, \ldots, \mathbf{B}_M\}$ of video clips, a sentential query $\mathbf{s}$, and a lexicon Λ, find the video clip $\mathbf{B}_{m^*}$ that best matches the query.

Using the Sentence Tracker for Retrieval

Given a set $\{\mathbf{B}_1, \ldots, \mathbf{B}_M\}$ of video clips, a sentential query $\mathbf{s}$, and a lexicon Λ, find the video clip $\mathbf{B}_{m^*}$ that best matches the query.

$$m^* = \arg\max_m \mathcal{S}(\mathbf{B}_m, \mathbf{s}, \Lambda)$$

Using the Sentence Tracker for Retrieval

Given a set $\{\mathbf{B}_1, \ldots, \mathbf{B}_M\}$ of video clips, a sentential query $\mathbf{s}$, and a lexicon Λ, find the video clip $\mathbf{B}_{m^*}$ that best matches the query.

$$m^* = \arg\max_m \mathcal{S}(\mathbf{B}_m, \mathbf{s}, \Lambda)$$

Barbu, A., Siddharth, N., and Siskind, J.M., *Language-driven video retrieval*, Vision Meets Cognition: Functionality, Physics, Intentionality and Causality, CVPR workshop, 2014.

Using the Sentence Tracker for Acquisition

Given a training set $\{(\mathbf{B}_1, \mathbf{s}_1), \ldots, (\mathbf{B}_M, \mathbf{s}_M)\}$ of video clips paired with sentential descriptions, learn the best lexicon Λ^*.

Using the Sentence Tracker for Acquisition

Given a training set $\{(\mathbf{B}_1, \mathbf{s}_1), \ldots, (\mathbf{B}_M, \mathbf{s}_M)\}$ of video clips paired with sentential descriptions, learn the best lexicon Λ^*.

$$\Lambda^* = \arg\max_{\Lambda} \prod_{m=1}^{M} \mathcal{S}(\mathbf{B}_m, \mathbf{s}_m, \Lambda)$$

Using the Sentence Tracker for Acquisition

Given a training set $\{(\mathbf{B}_1, \mathbf{s}_1), \ldots, (\mathbf{B}_M, \mathbf{s}_M)\}$ of video clips paired with sentential descriptions, learn the best lexicon Λ^*.

$$\Lambda^* = \arg\max_{\Lambda} \prod_{m=1}^{M} \mathcal{S}(\mathbf{B}_m, \mathbf{s}_m, \Lambda)$$

Yu, H. and Siskind, J.M., *Grounded language learning from video described with sentences*, ACL, pp. 56–63, 2013.

Live Demo

Live Demo

Live Demo

Live Demo

Outline

The Blessing

- existing object detection method: SVM over HOG (Dalal & Triggs, Felzenszwalb et al.)

- existing object detection method: SVM over HOG
  (Dalal & Triggs, Felzenszwalb et al.)
- existing action recognition method: time series modeled by HMM
  (trained with Baum-Welch)

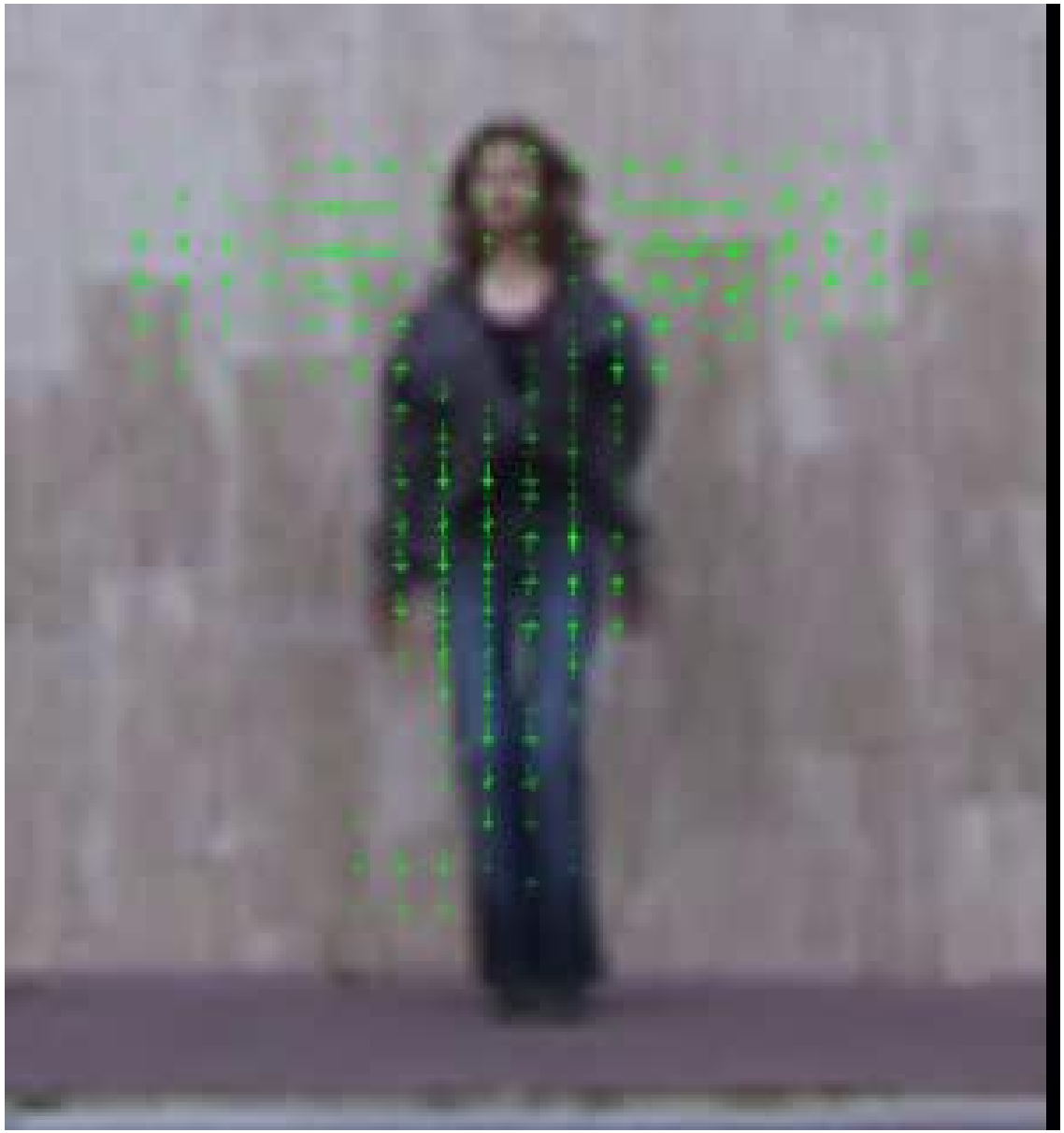
- existing object detection method: SVM over HOG
  (Dalal & Triggs, Felzenszwalb et al.)
- existing action recognition method: time series modeled by HMM
  (trained with Baum-Welch)
- idea: Felzenszwalb+Baum-Welch

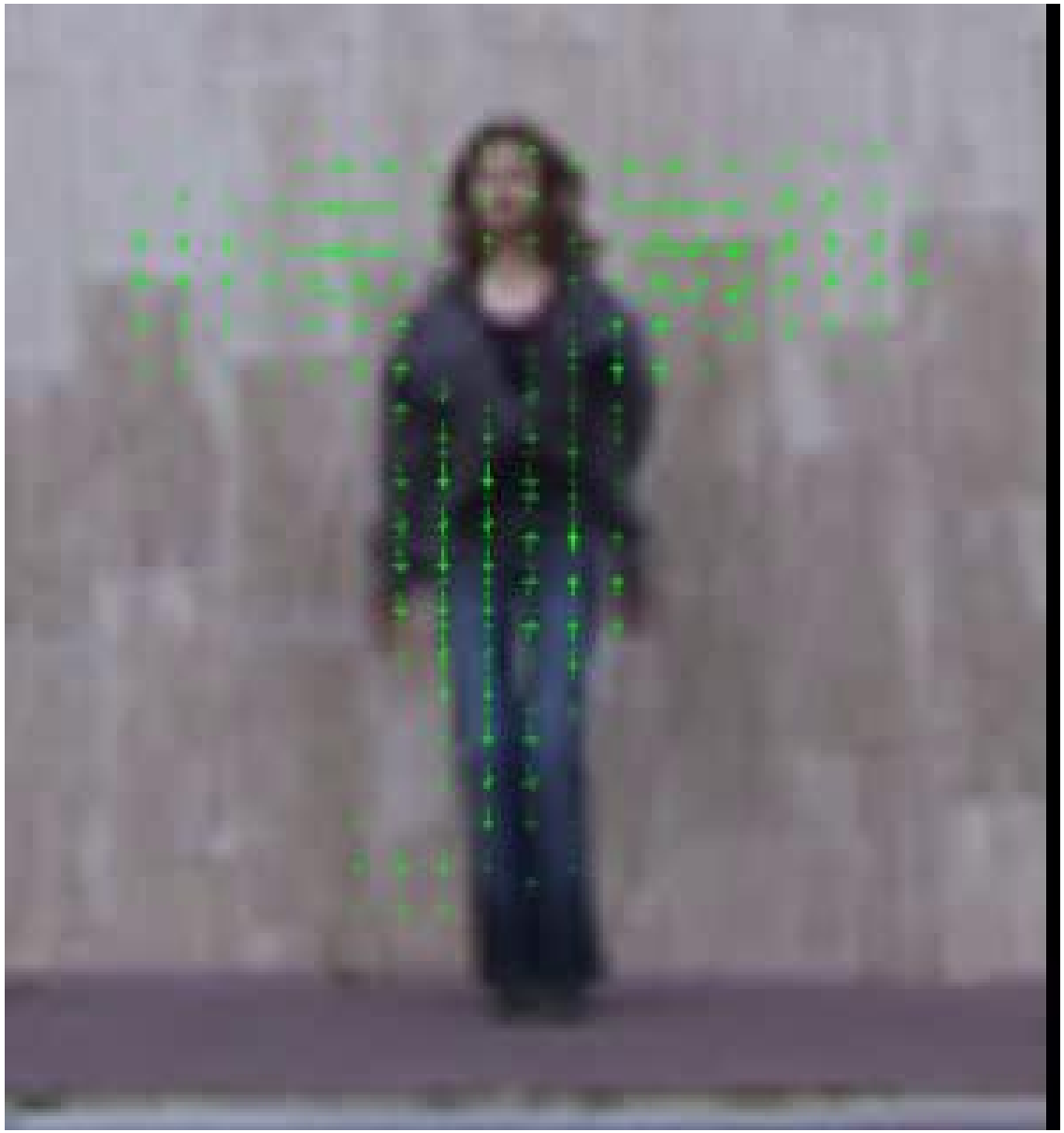
- existing object detection method: SVM over HOG
  (Dalal & Triggs, Felzenszwalb et al.)
- existing action recognition method: time series modeled by HMM
  (trained with Baum-Welch)
- idea: Felzenszwalb+Baum-Welch
- HMM with SVM output model

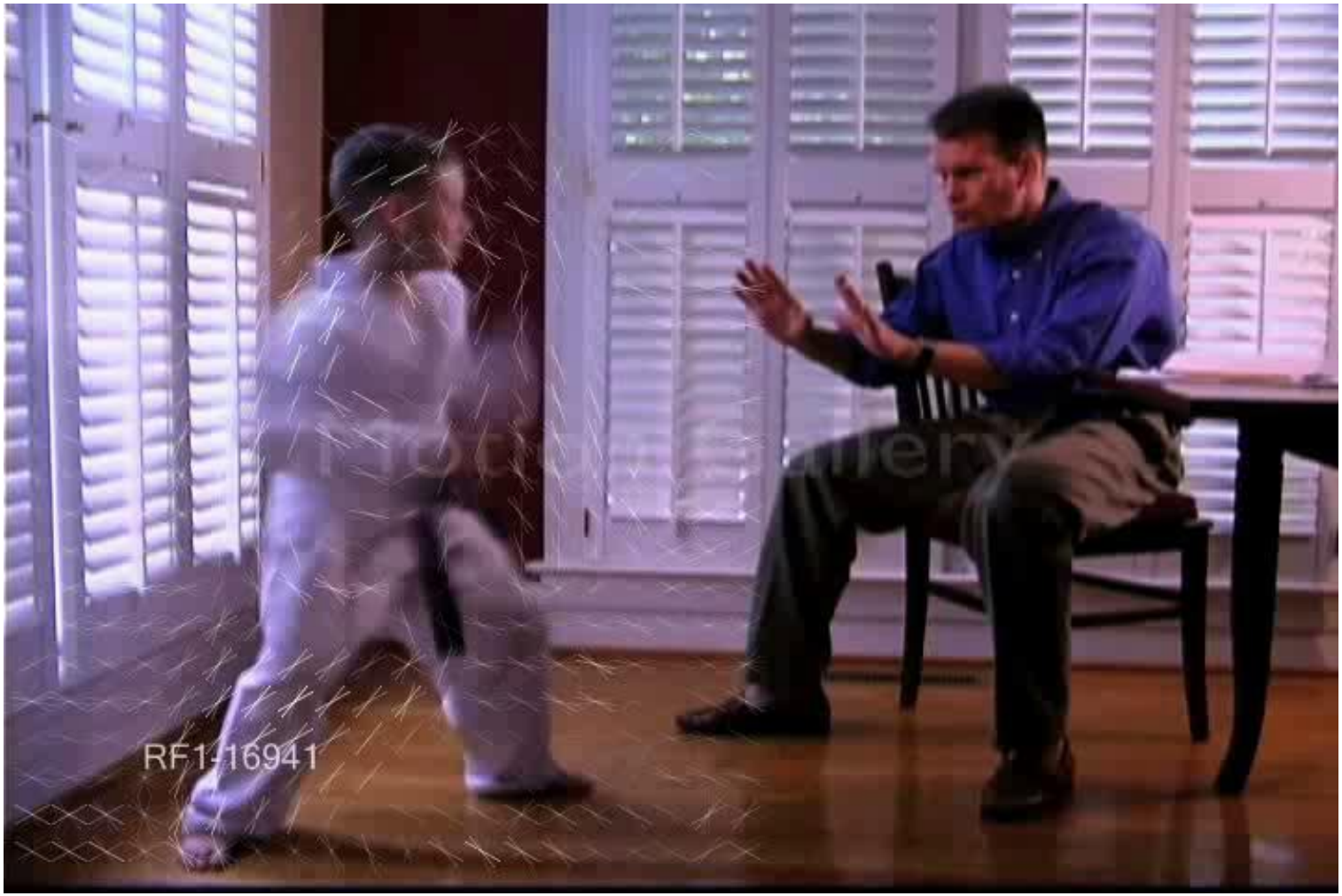
- existing object detection method: SVM over HOG
  (Dalal & Triggs, Felzenszwalb et al.)
- existing action recognition method: time series modeled by HMM
  (trained with Baum-Welch)
- idea: Felzenszwalb+Baum-Welch
- HMM with SVM output model
- SVM output model over HOG and HOF (retinotopic flow fields)

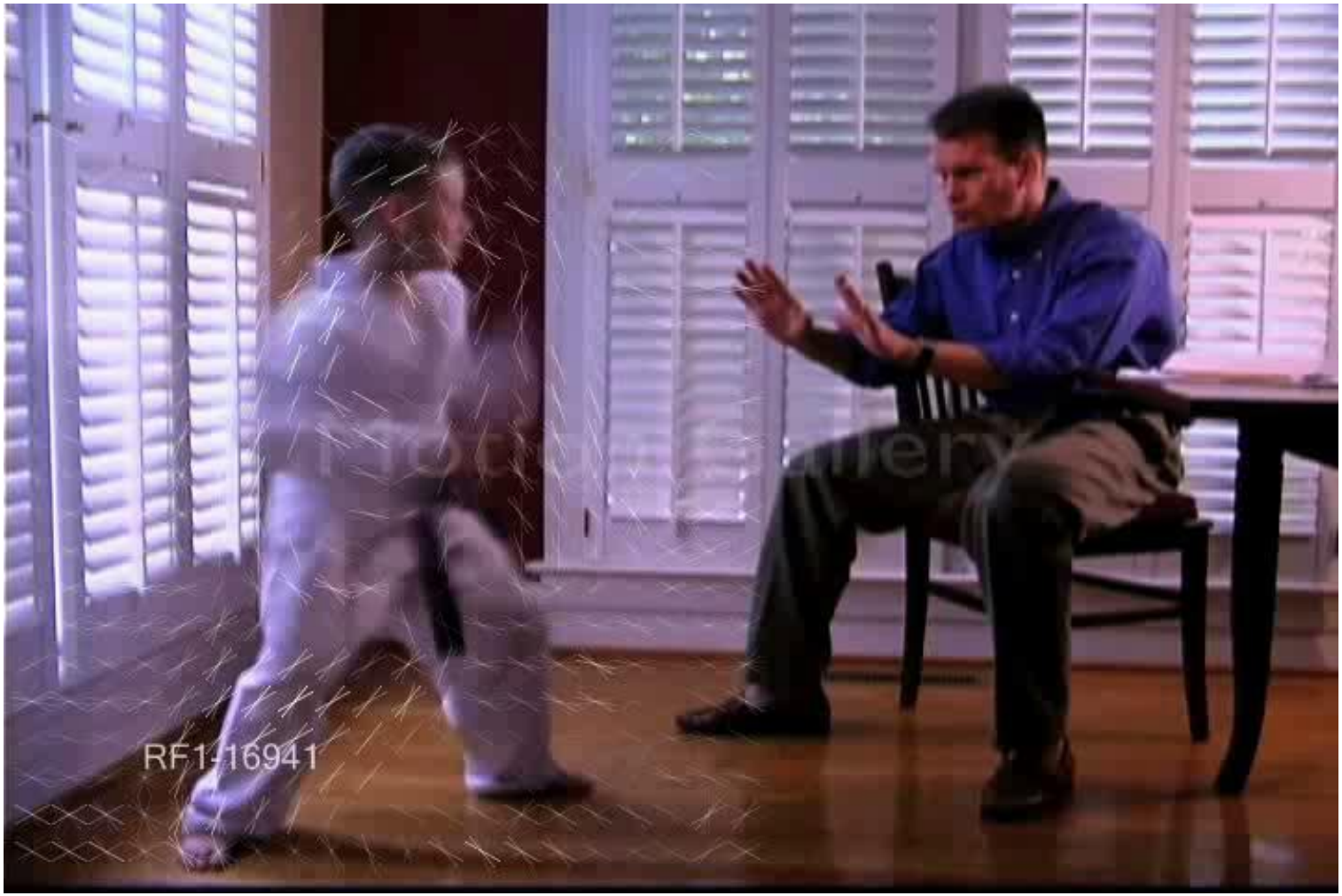
- existing object detection method: SVM over HOG
  (Dalal & Triggs, Felzenszwalb et al.)
- existing action recognition method: time series modeled by HMM
  (trained with Baum-Welch)
- idea: Felzenszwalb+Baum-Welch
- HMM with SVM output model
- SVM output model over HOG and HOF (retinotopic flow fields)
- train by gradient ascent over aggregate score of a labeled dataset

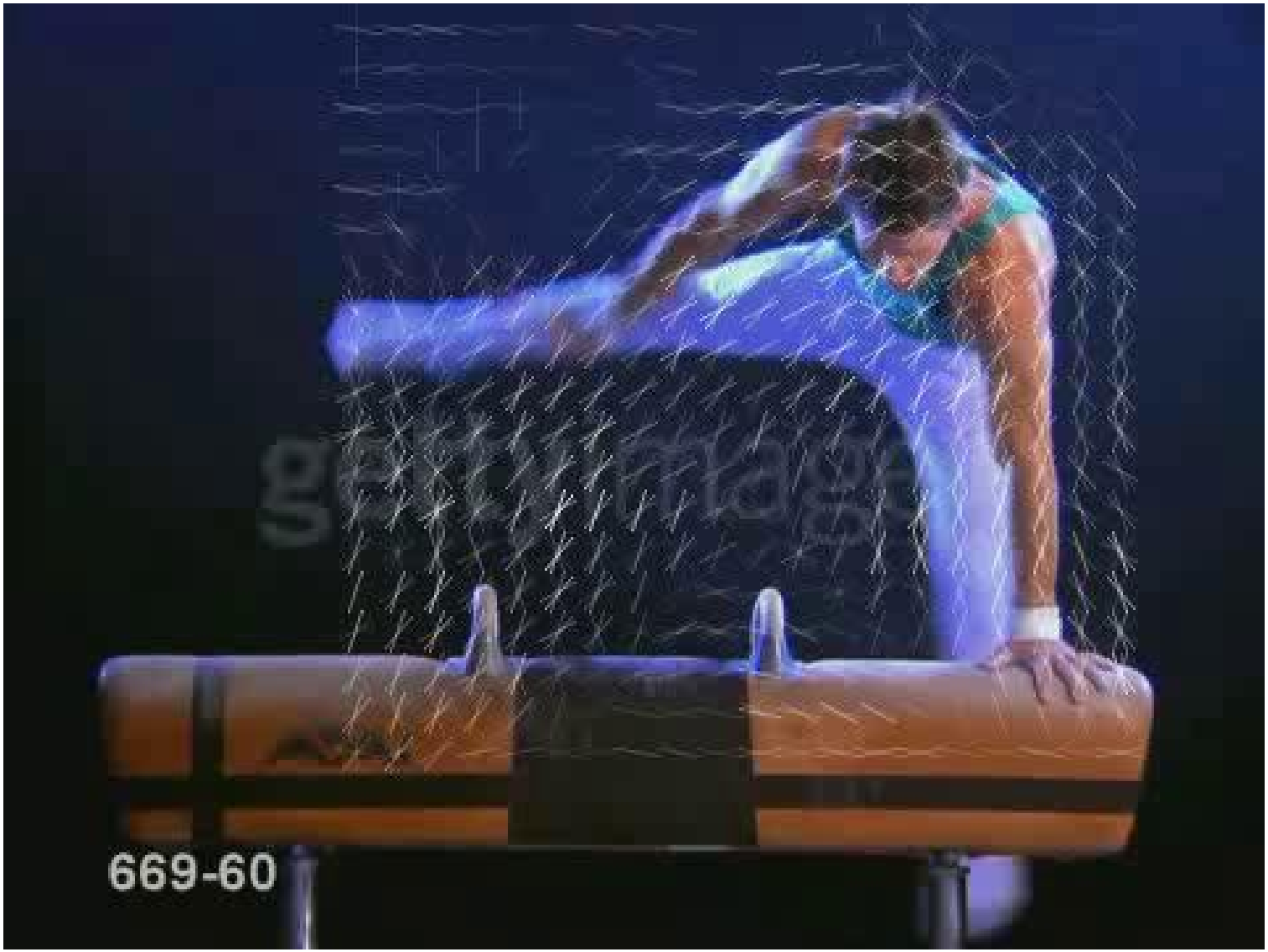
- existing object detection method: SVM over HOG
  (Dalal & Triggs, Felzenszwalb et al.)
- existing action recognition method: time series modeled by HMM
  (trained with Baum-Welch)
- idea: Felzenszwalb+Baum-Welch
- HMM with SVM output model
- SVM output model over HOG and HOF (retinotopic flow fields)
- train by gradient ascent over aggregate score of a labeled dataset
- classify by selecting class with highest score on unseen video

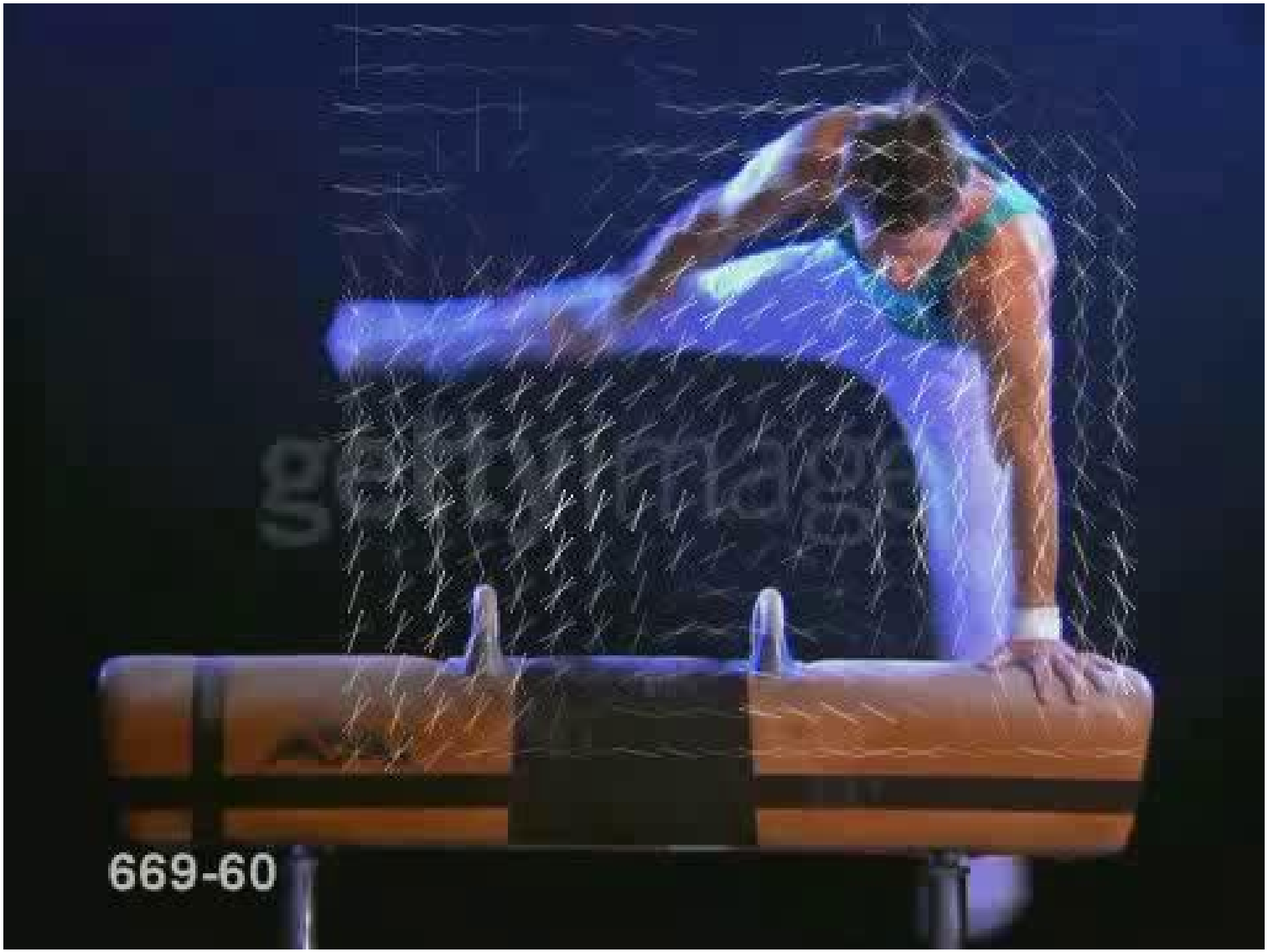


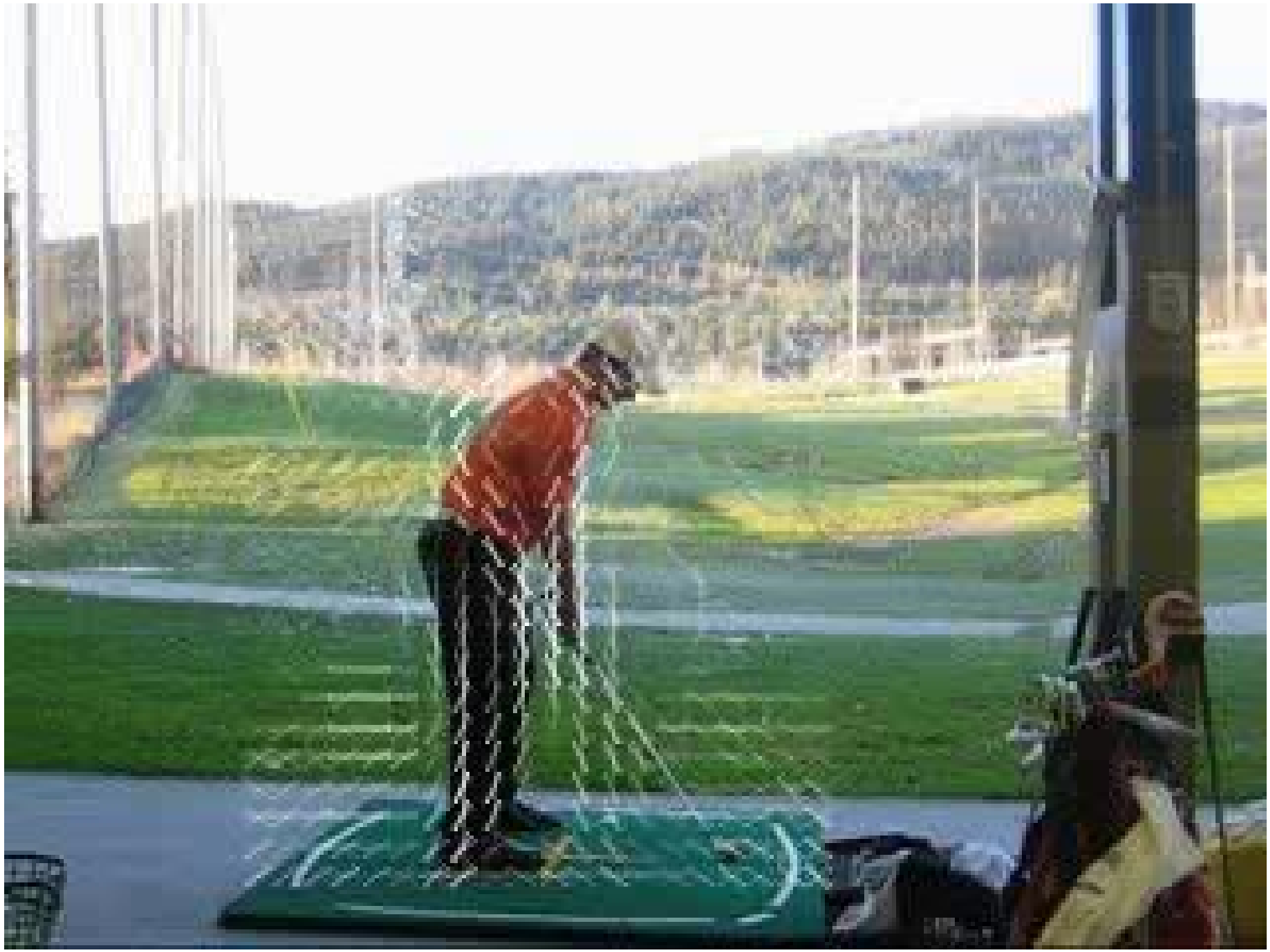


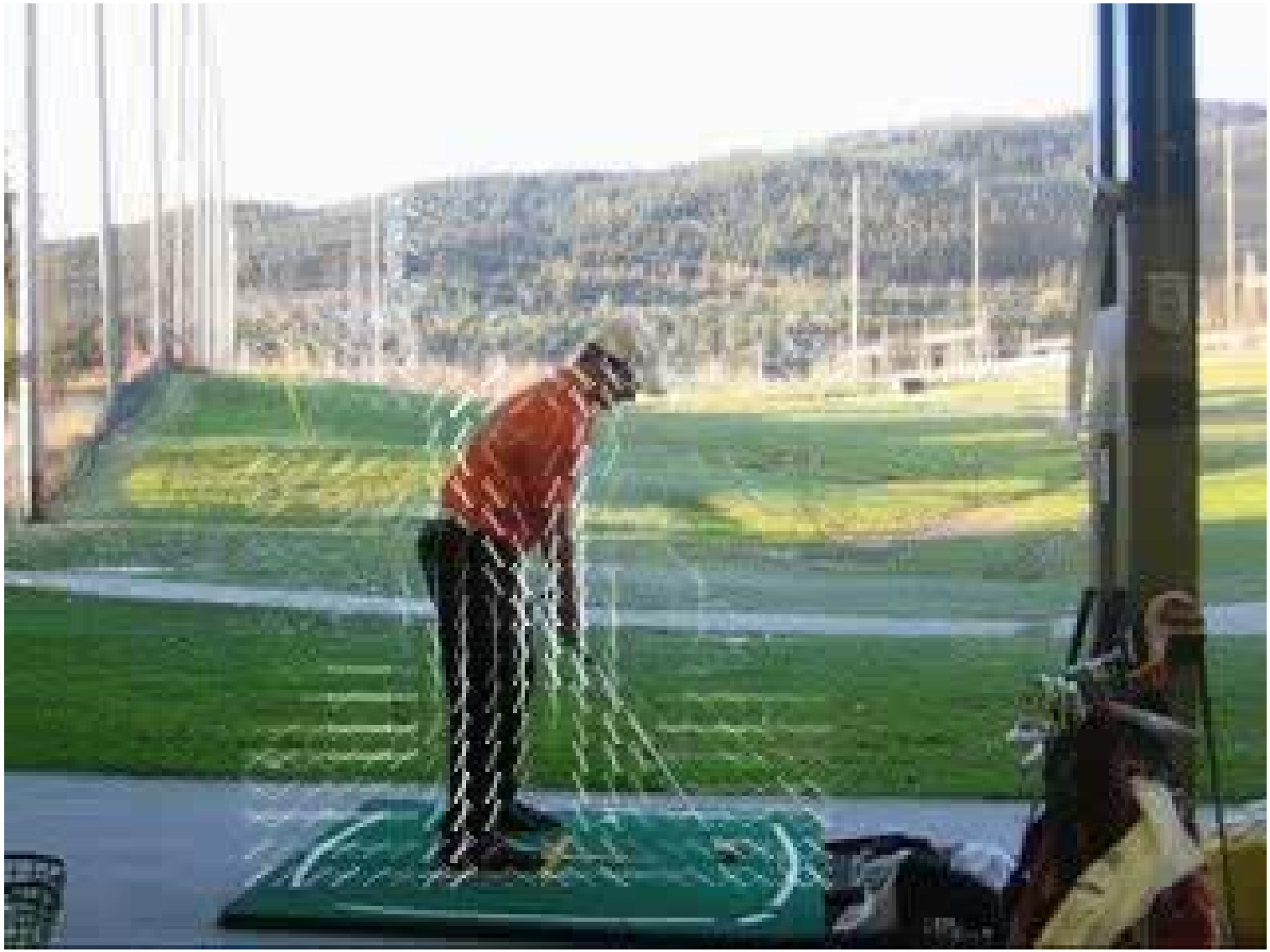

RF1-16941

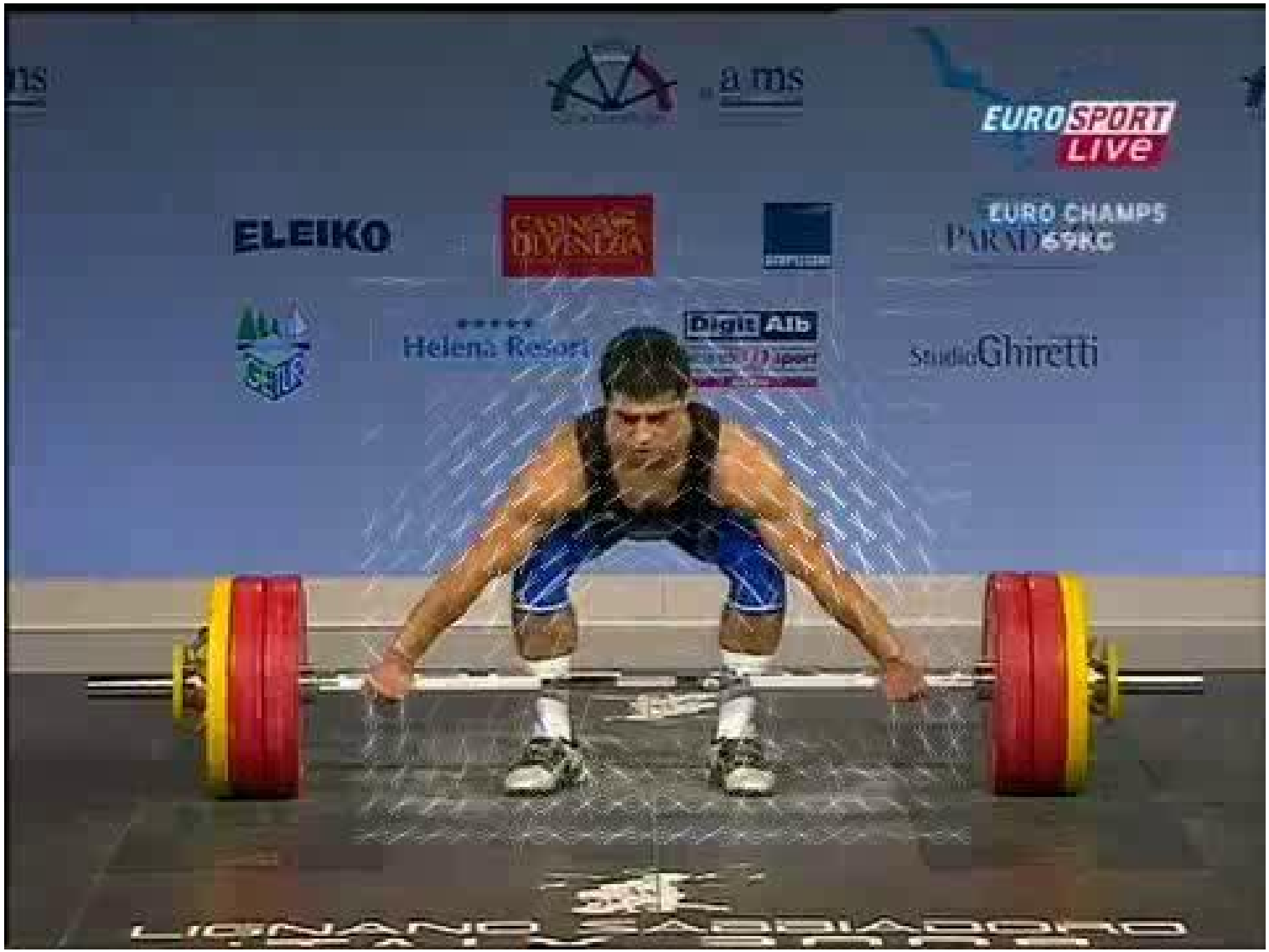

RF1-16941

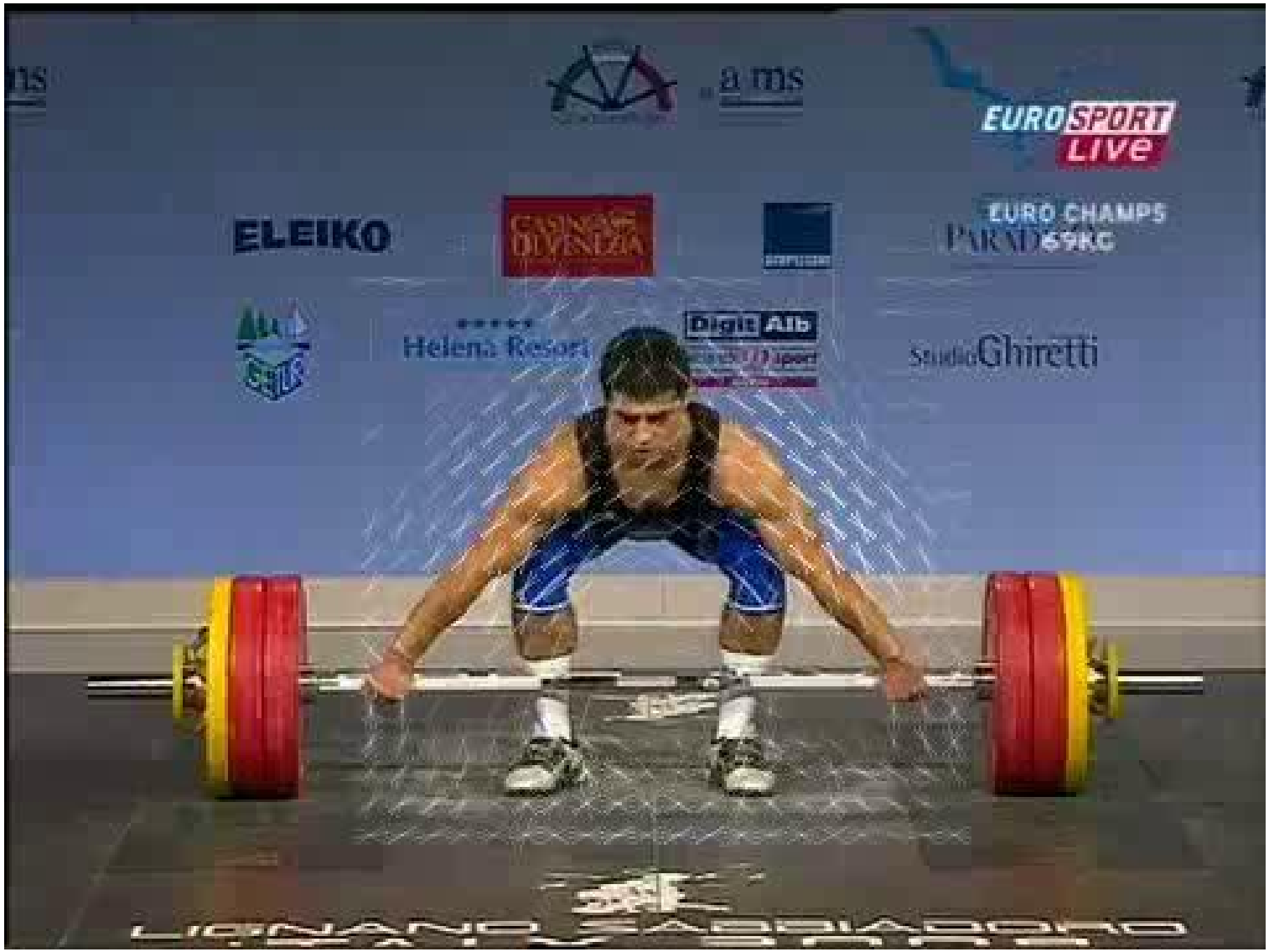

669-60

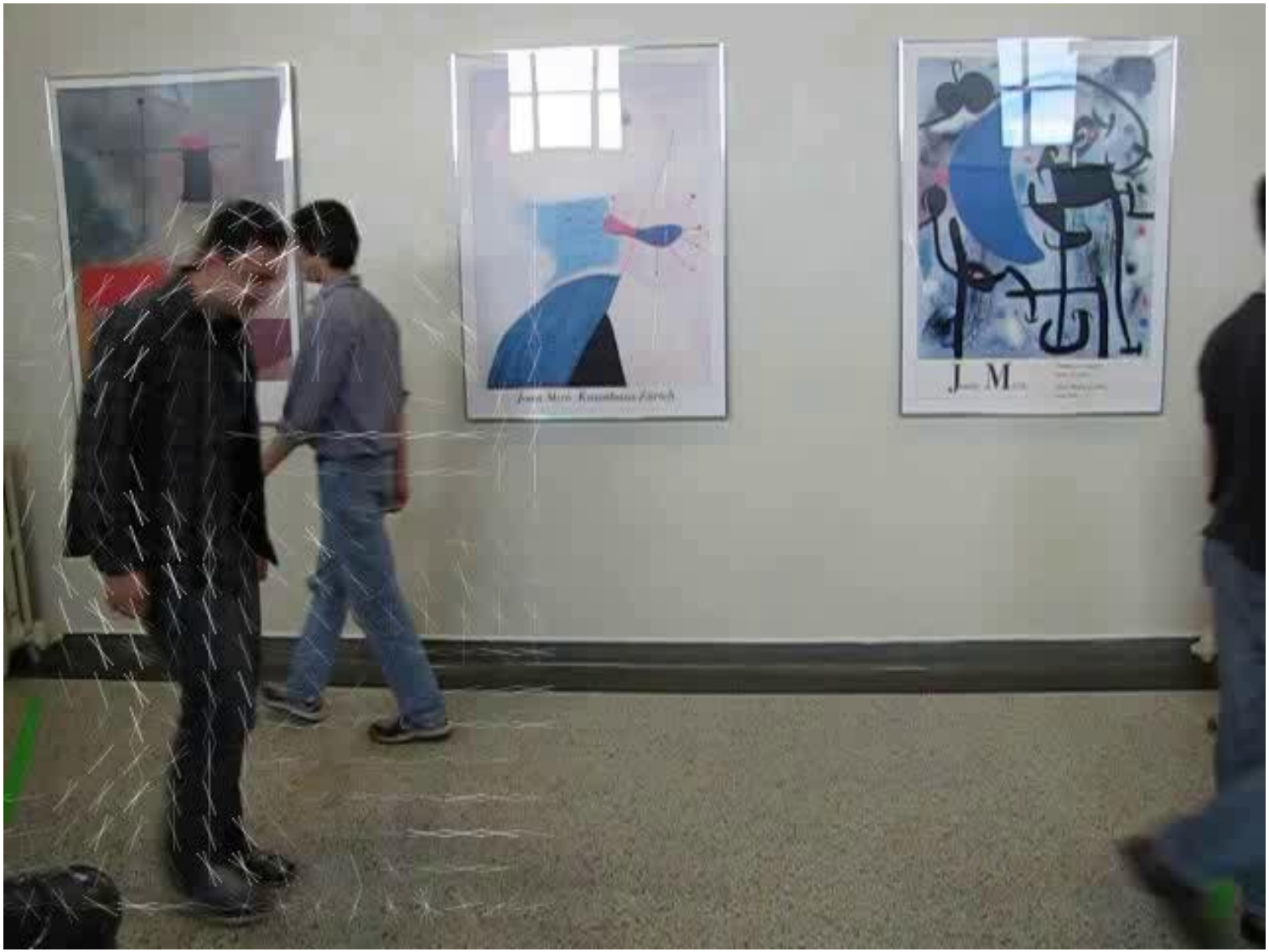

669-60

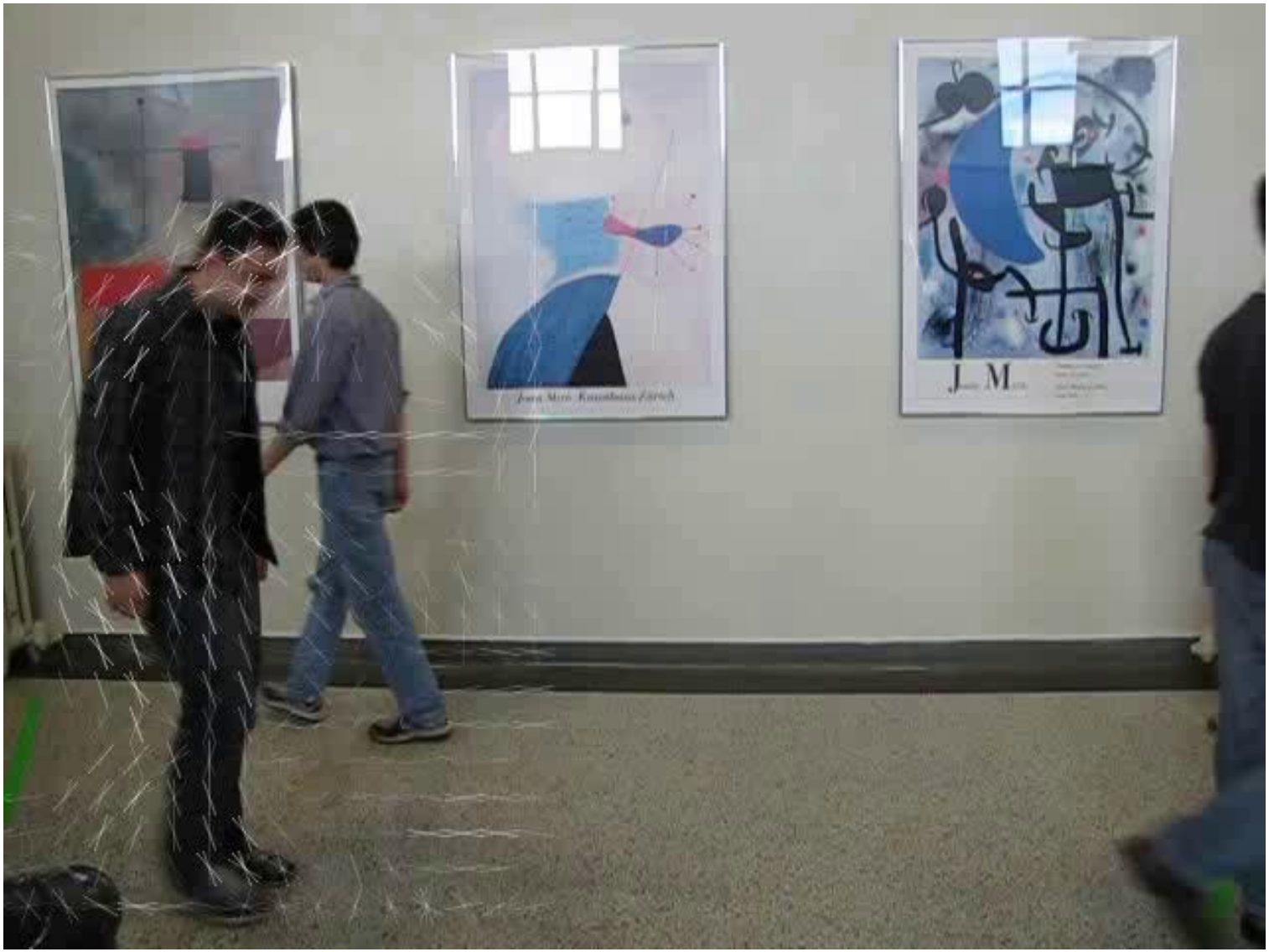


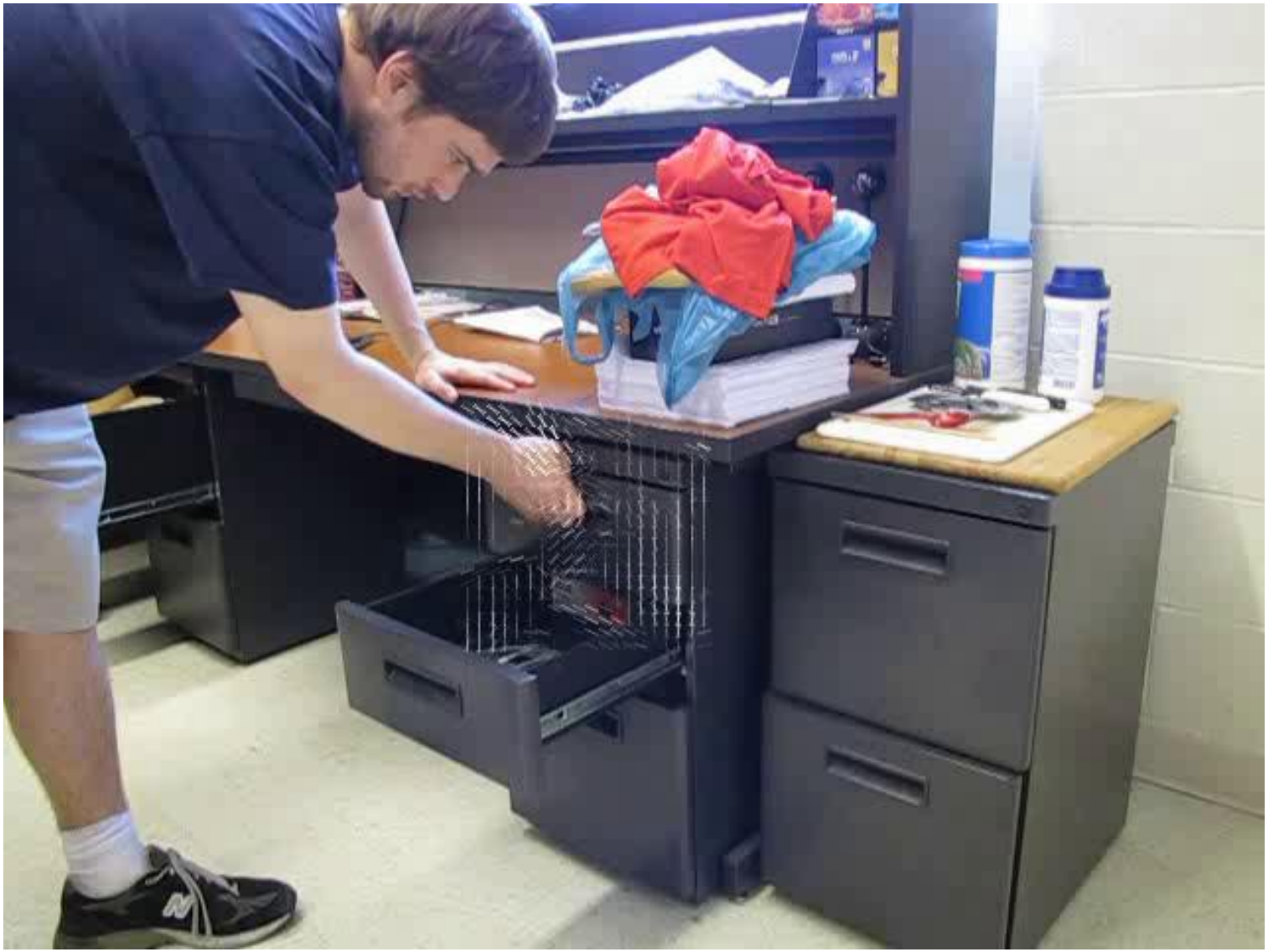


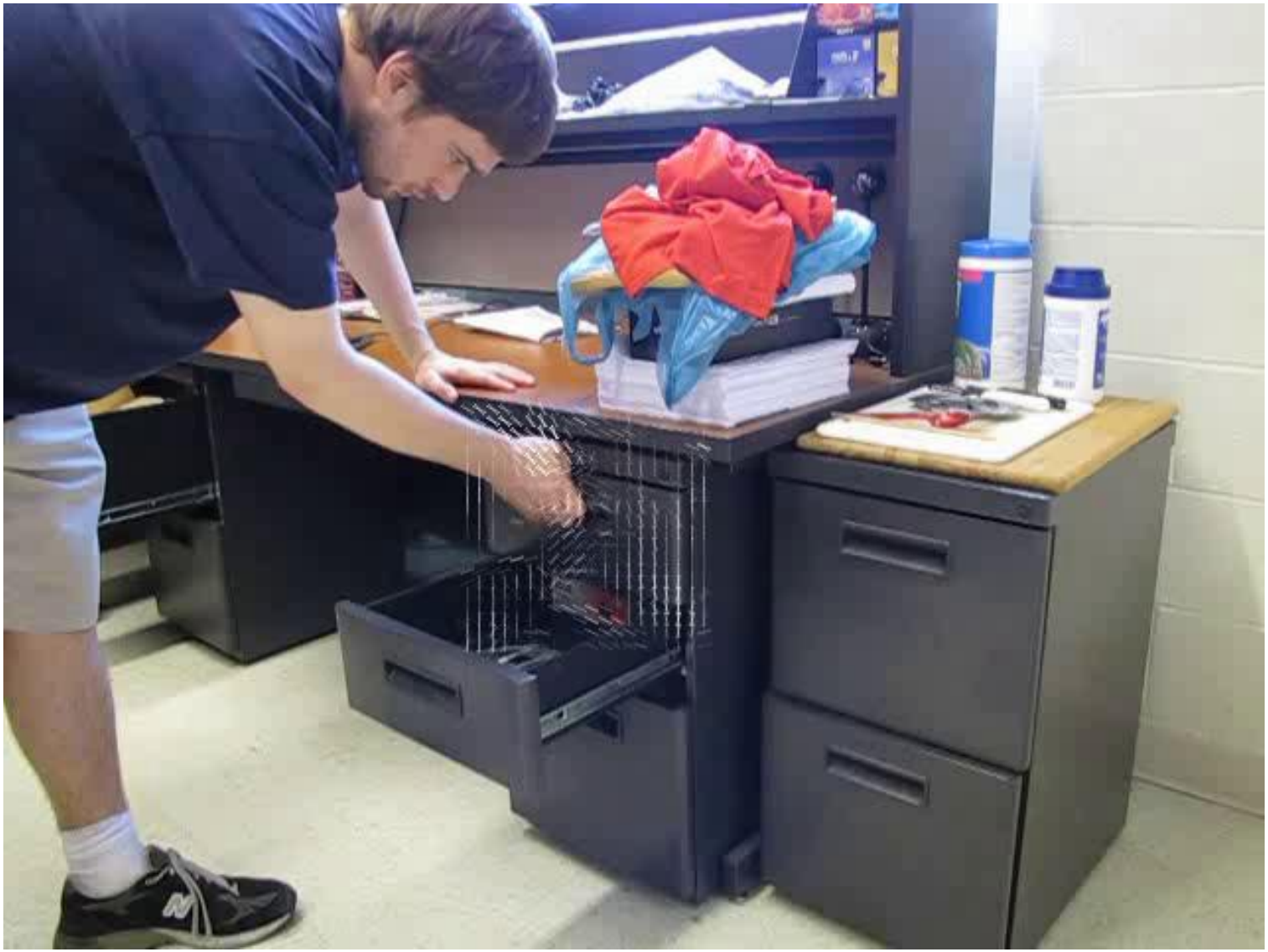

EUROSPORT
Live
ELEIKO
EURO CHAMPS
Helena Resort
Digit Alb
Studio Ghiretti


EUROSPORT
Live
ELEIKO
EURO CHAMPS
Helena Resort
Digit Alb
Studio Ghiretti


J. M


J. M





The Curse