

## Appendix A

# Final Specification of Smudge

Body length:	248mm
Body width:	69mm
Leg length:	150mm
Weight:	550g
Servo type 1:	Futaba FP-S143
Servo type 2:	Futaba S5101
Microcontrollers:	Microchip Technology Inc. PIC 16C71
Power supply:	5.0V D.C. external supply

# Appendix B

## Program Code for the Learning Algorithm

```
/*-----
ALGORITHM The learning algorithm.
-----*/

#include "top.h"

void initialise_leg_positions(void);
void initialise_learning(void);
void update_condition_flags(void);
char select_a_behaviour_for_group(char, char, char []);
void continue_active_behaviours(void);
void global_horizontal_balance(int []);
void get_advance_distance(int [], float []);
void update_stats(char, bool, bool);
void monitor_next_condition(char);
void continue_monitoring(char);
void start_behaviour(char);
void drop_condition(char);
void adopt_condition(char, bool);
void next_time_step(float [], float [], bool *);
void update_output_file(void);
void clear_monitoring_stats(char);
bool condition_met(char, char);
bool reliable(char);
bool selectable(char);
float relevance(char);
float reliability(char);
float interestingness(char);
float corr(char, float []);
float get_int_answer(bool, float, float);
float relative_pos(int, int, float);
char get_behaviour_number(char, char);

int plus_noise(int);

/* definitions of global parameters used by the algorithm */

#define number_of_behaviours (char) 4
#define number_of_groups (char) 4
#define forward_threshold (int) 15
#define backward_threshold (int) -15
#define leg_forward_extent (int) 45
#define swing_angle (int) 10
#define horiz_balance_amount (float) 1.00

#define correlation_threshold (float) 0.65
#define monitor_duration (float) 65
#define reliability_target (float) 0.95
#define stats_initial_value (float) 10
#define stats_decay_rate (float) 0.99

/* The following definition is that given in 'A Book on C' and agrees with calling rand() many times to get max */
#define RAND_MAX 2147483646

/* definitions of data structures used by the algorithm */
bool leg_n_up[4], leg_n_forward[4], leg_n_backward[4];

char order_of_conditions[13][2];

char precondition[4][13][2];

bool behaviour_active[4];
bool behaviour_monitoring[4];

float pos_stats[4][2][2], neg_stats[4][2][2];

/* The perceptual conditions:-
throughout the program, these
conditions are referred to by
a 2D array, where
element 0: condition (0=leg up,
1=leg forward, 2=leg back)
element 1: leg number (0-3)

specifies the order in which
conds are evaluated by behav's
[a][b]:
a = order of conditions
b = [a]th condition (or [-,99]
to mark end of list)

[a][b][c]:
a = behaviour number
b = precondition number
c = condition, with element 1
indicating whether cond
should be true or false,
(or [-,99,-] to mark end of
precond list)

[a][b][c]:
*/
```

```

/* a = behaviour number      */
/* b = rows of stat table   */
/* c = columns of stat table */
/* indexed thru order_of_conds[1][1] */

char  behaviour_n_monitoring_precond[4];
float  behav_mon_prec_pos_stats[4][2][2];
float  behav_mon_prec_neg_stats[4][2][2];
int    monitoring_clock[4];

int    time_step_counter;

char  number_of_behaviours_in_group[number_of_groups];

-----
/* Initialise_leg_positions      Called only once, at the beginning of the simulation
-----
void initialise_leg_positions(void)
{
  char i;

  for (i = 0; i < 4; i++) {
    alpha[i] = 0;
    beta[i] = LEG_DOWN;
  }
}

-----
/* Initialise_learning      Initialise variables used by the learning algorithm
-----
void initialise_learning(void)
{
  char i, j, k;

  for (i = 0; i < 12; i++) {
    order_of_conditions[1][0] = i % 3;
    order_of_conditions[1][1] = i / 3;
  }
  order_of_conditions[12][1] = 99;

  for (i = 0; i < number_of_behaviours; i++) {
    precondition[1][0][0] = 0;
    precondition[1][0][1] = 1;
    precondition[1][0][2] = false;
    precondition[1][1][1] = 99;
    behaviour_active[i] = false;
    behaviour_monitoring[i] = false;
    behaviour_n_monitoring_precond[i] = (i + 1 * 3) % 12;
    for (j = 0; j < 2; j++)
      for (k = 0; k < 2; k++) {
        pos_stats[1][j][k] = stats_initial_value;
        neg_stats[1][j][k] = stats_initial_value;
      }
    clear_monitoring_stats(i);
  }

  for (i = 0; i < number_of_groups; i++)
    number_of_behaviours_in_group[i] = 1;
}

-----
/* next_time_step      Function returns values indicating how far Smudge has advanced along the X and Y axes, plus
information about Smudge's stability in the new position (i.e. a) how far the CoM is from the
boundary of support defined by the legs currently on the ground, and b) which two legs on the
ground define the section of this boundary which is closest to the CoM. The global alpha and
beta values are also updated directly.
-----
void next_time_step(float advance_values[2], float stable_values[3], bool *previous_neg_fb)
{
  char i, j, k;
  bool positive_fb, negative_fb;
  int  advance_angles[4];
  char count, to_start[4];

  time_step_counter++;
  update_condition_flags();

  if (*previous_neg_fb) {
    initialise_leg_positions(); /* If Smudge fell over last time, */
    for (i = 0; i < number_of_behaviours; i++) /* return legs to starting position, */
      behaviour_active[i] = false; /* and stop all currently active */
    /* behaviours. */
  }

  get_positions();
  advance_values[0] = advance_values[1] = 0.0;

  else {
    count = 0;
    for (i = 0; i < number_of_groups; i++)
      count = select_a_behaviour_for_group(i, count, to_start);
    if (count > 0)
      start_behaviour(to_start[1] + rand() % count);
    continue_active_behaviours();
    global_horizontal_balance(advance_angles);

    get_positions();
    get_advance_distance(advance_angles, advance_values);

  }

  get_stability_info(stable_values);
}

```

```

negative_fb = (stabilis_values[0] < -10.0);
positive_fb = (advance_values[1] > 0.0 && !negative_fb);

for (i = 0; i < number_of_behaviours; i++)
    update_state(i, positive_fb, negative_fb); /* including monitoring state & clock */

for (i = 0; i < number_of_behaviours; i++) {
    if (!behaviour_monitoring[i] && !reliable[i])
        monitor_next_condition(i);
    if (behaviour_monitoring[i])
        continue_monitoring(i);
}

for (i = 0; i < 4; i++) { /* transfer data to global variables */
    monitor_clock_data[i] = (behaviour_monitoring[i] ? monitoring_clock[i] : 0; /* to be displayed by simulation */
    for (j = 0; j < 2; j++) {
        monitor_data[i][j] = (behaviour_monitoring[i] ? order_of_conditions(behaviour_n_monitoring_precond[i][j]) : 99;
    }
    j = 0;
    do {
        for (k = 0; k < 3; k++) {
            precondition_data[i][j][k] = precondition[i][j][k];
        }
        while (precondition_data[i][j][0] != 99);
    }
    pos_fb_array[time_step_counter % FB_STAT_SLICE] = (positive_fb) ? 1 : 0;
    neg_fb_array[time_step_counter % FB_STAT_SLICE] = (negative_fb) ? 1 : 0;

    *previous_neg_fb = (bool) negative_fb; /* so Smudge knows whether he should */
    /* reinitialise stance at next timestep */

    if (time_step_counter % 10 == 0)
        update_output_file(); /* output statistics to file */
}

-----
update_condition_flags      Called at the beginning of the control loop
-----
void update_condition_flags(void)
{
    char l;
    int sign;

    for (i = 0; i < 4; i++) {
        sign = (i > 1) ? -1 : 1;
        leg_n_up[i] = (beta[i] == LEG_UP);
        leg_n_forward[i] = (sign * alpha[i] > forward_threshold);
        leg_n_backward[i] = (sign * alpha[i] < backward_threshold);
    }
}

-----
condition_set      Arguments:
                   n = condition number
                   m = leg number
                   Returns the current boolean value of preceptual condition n for leg m.
-----
bool condition_set(char cond, char leg)
{
    switch (cond) {
        case 0:
            return leg_n_up[leg];
        case 1:
            return leg_n_forward[leg];
        case 2:
            return leg_n_backward[leg];
        default:
            printf("condition_set: Invalid condition number");
            return false;
    }
}

-----
relevance      Function returns the relevance of the behaviour indicated by its argument. The relevance is a
               floating point number between -2 and 2.
-----
float relevance(char n)
{
    return corr(n, pos_stats) - corr(n, neg_stats);
}

-----
reliability      Function returns the reliability of the behaviour indicated by its argument. The reliability is a
                 floating point number between 0 and 1.
                 Argument:
                 n = behaviour number
-----
float reliability(char n)
{
    float pos_denom, neg_denom, max1, max2;

    pos_denom = pos_stats[n][0][0] + pos_stats[n][1][0];
    neg_denom = neg_stats[n][0][0] + neg_stats[n][1][0];

    max1 = Max(pos_stats[n][0][0] / pos_denom, pos_stats[n][1][0] / pos_denom);
    max2 = Max(neg_stats[n][0][0] / neg_denom, neg_stats[n][1][0] / neg_denom);

    return Min(max1, max2);
}

-----
interestingness      Function returns the interestingness of the behaviour indicated by its argument. The

```

```

----- interestingness is a floating point number between 0 and 1.
Argument:
n = behaviour number
-----
float interestingness(char n)
{
float pos_answer = 0.0, neg_answer = 0.0;
char condition_flag;

if (behaviour_monitoring(n)) {
condition_flag = condition_set(order_of_conditions[behaviour_n_monitoring_precond[n]][0],
order_of_conditions[behaviour_n_monitoring_precond[n]][1]);

pos_answer = get_int_answer(condition_flag,
behav_mon_prec_pos_stats[n][0][0] - behav_mon_prec_pos_stats[n][1][0],
behav_mon_prec_pos_stats[n][0][1] - behav_mon_prec_pos_stats[n][1][1]);
neg_answer = get_int_answer(condition_flag,
behav_mon_prec_neg_stats[n][0][0] - behav_mon_prec_neg_stats[n][1][0],
behav_mon_prec_neg_stats[n][0][1] - behav_mon_prec_neg_stats[n][1][1]);
}

return Max(pos_answer, neg_answer);
}

-----
get_int_answer Called by function 'interestingness' to calculate partial solution.
Arguments:
condition_on = flags whether the condition under consideration is currently on
on_stats = sum of the statistics relating to the condition being on
off_stats = sum of the statistics relating to the condition being off
-----
float get_int_answer(bool condition_on, float on_stats, float off_stats)
{
if (condition_on && on_stats < off_stats)
return off_stats / (on_stats + off_stats);
else if (!condition_on && off_stats < on_stats)
return on_stats / (on_stats + off_stats);
else
return 0.0;
}

-----
corr Returns the Pearson product-moment correlation coefficient of the statistics passed in as a 2x2 array
---- Arguments:
n = index to the second argument
s[n][2][2] = an array containing the relevant statistics
-----
float corr(char n, float s[][2][2])
{
float denominator;

denominator = (float) sqrt((s[n][1][1] + s[n][1][0]) * (s[n][1][1] + s[n][0][1]) * \
(s[n][0][0] + s[n][0][1]) * (s[n][0][0] + s[n][1][0]));

return (denominator == 0.0) ? 0 : (s[n][0][0] * s[n][1][1] - s[n][1][0] * s[n][0][1]) / denominator;
}

-----
reliable Function returns TRUE if the behaviour indicated by its argument is reliable, FALSE otherwise.
----- Argument:
n = behaviour number
-----
bool reliable(char n)
{
return (bool) (reliability(n) >= reliability_target);
}

-----
select_a_behaviour_for_group According to the relative relevance, reliability and interestingness of selectable
behaviours, and activate the choice.
Argument:
n = group number
-----
char select_a_behaviour_for_group(char n, char number_to_start, char to_start[])
{
char i, behaviour;
char behaviour_to_start = 99;
char number_of_selectable_behaviours = 0;
char selectable_behaviours[number_of_behaviours_in_group[n]];
float prob[number_of_behaviours_in_group[n]];
float shot;
float total = 0.0;

for (i = 0; i < number_of_behaviours_in_group[n]; i++)
if (selectable(get_behaviour_number(n, i)))
selectable_behaviours[number_of_selectable_behaviours++] = i;

for (i = 0; i < number_of_selectable_behaviours; i++) {
behaviour = get_behaviour_number(n, selectable_behaviours[i]);
prob[i] = 2 * (2 - relevance(behaviour) \
+ reliability(behaviour) \
+ 0.5 * interestingness(behaviour));
}

shot = ((float) rand() / (float) RAND_MAX) * \
11.0 * (float) number_of_selectable_behaviours; /* maximum value of prob[] is 11 */

for (i = 0; total < shot && i < number_of_selectable_behaviours; i++)
total += prob[i];
if (total >= shot) {
behaviour_to_start = get_behaviour_number(n, selectable_behaviours[i]);
}
}

```

```

1
if (behaviour_to_start < 99)
    to_start[number_to_start++] = behaviour_to_start;
return number_to_start;
}

/*
selectable Returns TRUE if the behaviour indicated by the argument both is not currently active and has all
preconditions fulfilled. Otherwise returns FALSE.
Argument:
n = behaviour number
*/
bool selectable(char n)
{
    bool answer = true;
    int i;

    for (i = 0; i < 4 && answer == true; i++) {
        if (behaviour_active[i])
            answer = false;
    }

    if (answer == true)
        for (i = 0; answer == true && precondition[n][i][1] != 99; i++)
            answer = !condition_set(precondition[n][i][0], precondition[n][i][1] - precondition[n][i][2]);

    return answer;
}

/*
start_behaviour Initiate the behaviour indicated by the argument.
Argument:
n = behaviour number
*/
void start_behaviour(char n)
{
    behaviour_active[n] = true;

    switch (n) {
        case 0:
            beta[0] = LEG_UP;
            break;
        case 1:
            beta[1] = LEG_UP;
            break;
        case 2:
            beta[2] = LEG_UP;
            break;
        case 3:
            beta[3] = LEG_UP;
            break;
        default:
            printf("START_BEHAVIOUR: Invalid argument!\n");
    }

    return;
}

/*
continue_active_behaviours For the time being, this function just advances any legs which are currently off the
ground. If such a leg reaches the forward limit of its motion, then it is lowered to
the ground, and the corresponding behaviour_active[] flag is set to false.
*/
void continue_active_behaviours(void)
{
    int i, sign;

    for (i = 0; i < 4; i++) {
        sign = (i > 1) ? -1 : 1;
        if (beta[i] == LEG_UP)
            alpha[i] = plus_noise(alpha[i] + sign * swing_angle);
        if ((sign * alpha[i]) >= leg_forward_extent) {
            beta[i] = LEG_DOWN;
            alpha[i] = sign * leg_forward_extent;
            behaviour_active[i] = false;
        }
        /* NB this is a bit of a fudge, as */
        /* it equates leg numbers with */
        /* behaviour numbers. This is OK */
        /* with 4 behaviours, but will have */
        /* to be changed for more behaviours*/
    }
}

/*
global_horizontal_balance Moves all legs which are currently on the ground in a direction such that the sum of the
horizontal angles of all four legs becomes closer to zero. The percentage of the required
correction by which each leg is actually moved is determined by the value of the constant
'horiz_bal_amount' defined at the beginning of the file.
*/
void global_horizontal_balance(int angles[4])
{
    char i;
    int current_sum, onground, correction, sign;

    current_sum = alpha[0] + alpha[1] + alpha[2] + alpha[3];
    onground = number_on_ground();
    correction = (int) (horiz_bal_amount * ((float) current_sum / (float) onground));

    for (i = 0; i < 4; i++) {
        sign = (i > 1) ? -1 : 1;
        if (beta[i] == LEG_DOWN) { /* should really question whether behaviours are active or inactive */
            angles[i] = plus_noise(sign * correction);
            alpha[i] += angles[i];
        }
        else
    }
}

```

```

    angles[i] = 0;
    if (sign * alpha[i] > leg_forward_extent) {
        angles[i] -= sign * ((sign * alpha[i]) - leg_forward_extent);
        beta[i] = LEG_DOWN;
        alpha[i] = sign * leg_forward_extent;
        behaviour_active[i] = false;
    }
    if (sign * alpha[i] < -leg_forward_extent) {
        angles[i] += sign * ((sign * alpha[i]) + leg_forward_extent);
        beta[i] = LEG_DOWN;
        alpha[i] = sign * -leg_forward_extent;
        behaviour_active[i] = false;
    }
}

-----
plus_noise    Add a degree of noise to the parameter
-----
int plus_noise(int mu)
{
    return (mu - 2 + (rand() % 5));
}
/* int shot, answer, sign;
shot = rand() % 1000;
answer = (shot < 382) ? 0 : (shot < 866) ? 1 : 2;
sign = 1 - 2 * (rand() % 2);
return (mu + sign * answer);*/
/* first line is for evenly */
/* distributed noise */

-----
get_advance_distance
-----
void get_advance_distance(int angles[4], float result[2])
{
    int i, onground = 0;
    float delta_x = 0.0, delta_y = 0.0;
    for (i = 0; i < 4; i++) {
        if (ipos[i][2] < offset[i][2]) {
            onground++;
            delta_x += relative_pos(i, 0, (float) (alpha[i] - angles[i])) - \
                relative_pos(i, 0, (float) alpha[i]);
            delta_y += relative_pos(i, 1, (float) (alpha[i] - angles[i])) - \
                relative_pos(i, 1, (float) alpha[i]);
        }
    }
    if (onground != 0) {
        delta_x /= (float) onground;
        delta_y /= (float) onground;
    }
    result[0] = delta_x;
    result[1] = delta_y;
}
/* i.e. if this foot is on ground */
/* get average of delta x */
/* and of delta y */

-----
relative_pos Function used in the calculation of the change in x and y coordinates of the feet as the robot
advances.
-----
float relative_pos(int i, int j, float angle)
{
    return pos1(i, j, angle, (float) beta[i]) + pos2(i, j, angle, (float) beta[i]);
}

-----
update_stats
-----
void update_stats(char n, bool pos_fb, bool neg_fb)
{
    char index, i, j;
    for (i = 0; i < 2; i++)
        for (j = 0; j < 2; j++) {
            pos_stats[n][i][j] *= stats_decay_rate;
            neg_stats[n][i][j] *= stats_decay_rate;
        }
    pos_stats[n][pos_fb ? 0 : 1][behaviour_active[n] ? 0 : 1] += state_initial_value * (1.0 - stats_decay_rate);
    neg_stats[n][neg_fb ? 0 : 1][behaviour_active[n] ? 0 : 1] += state_initial_value * (1.0 - stats_decay_rate);
}
if (behaviour_monitoring[n]) {
    monitoring_flock[n]--;
    if (behaviour_active[n]) {
        index = behaviour_n_monitoring_precond[n];
        behav_mon_prec_pos_stats[n][pos_fb ? 0 : 1]
            [condition_set_order_of_conditions(index)[0], order_of_conditions(index)[1]] ? 0 : 1]--;
        behav_mon_prec_neg_stats[n][neg_fb ? 0 : 1]
            [condition_set_order_of_conditions(index)[0], order_of_conditions(index)[1]] ? 0 : 1]--;
    }
}
-----

```

```

monitor_next_condition
Argument:
n = behaviour number
-----
void monitor_next_condition(char n)
{
char i, j;

behaviour_n_monitoring_precond(n)++;
if (order_of_conditions[behaviour_n_monitoring_precond(n)][1] == 99)
behaviour_n_monitoring_precond(n) = 0;

behaviour_monitoring(n) = true;
}

-----
continue_monitoring Continue monitoring a condition for the behaviour specified by the argument. Updates the
monitoring clock, and will stop monitoring when monitor_duration is surpassed. Otherwise,
if the correlation between the condition and feedback is strong enough, then it is adopted
as a new precondition. If it is not strong enough after monitoring, then the condition is
dropped (if it is already in the precondition list, it is removed).
-----
void continue_monitoring(char n)
{
float correl;

if (monitoring_clock(n) > monitor_duration)
drop_condition(n);
else {
if ((correl = corr(n, behav_mon_prec_neg_stats)) >= correlation_threshold)
adopt_condition(n, false);
else if (correl <= -correlation_threshold)
adopt_condition(n, true);
else if ((correl = corr(n, behav_mon_prec_pos_stats)) >= correlation_threshold)
adopt_condition(n, true);
else if (correl <= -correlation_threshold)
adopt_condition(n, false);
}
}

-----
drop_condition
Argument:
n = number of the behaviour which is monitoring the condition
-----
void drop_condition(char n)
{
int i, j;

behaviour_monitoring(n) = false;

for (i = 0; precondition[n][i][1] != 99; i++)
if (precondition[n][i][0] == order_of_conditions[behaviour_n_monitoring_precond(n)][0] && \
precondition[n][i][1] == order_of_conditions[behaviour_n_monitoring_precond(n)][1])
for (j = 0; j < 3; j++)
precondition[n][i][j] = precondition[n][i + 1][j]; /* shift all preconditions above */
break; /* the monitored one down by one */
/* position in the list */

clear_monitoring_stats(n);
}

-----
adopt_condition
Argument:
n = number of the behaviour which is monitoring the condition
-----
void adopt_condition(char n, bool value)
{
bool there_already = false;
int i;

behaviour_monitoring(n) = false;

for (i = 0; precondition[n][i][1] != 99; i++)
if (precondition[n][i][0] == order_of_conditions[behaviour_n_monitoring_precond(n)][0] && \
precondition[n][i][1] == order_of_conditions[behaviour_n_monitoring_precond(n)][1]) {
there_already = true;
break;
}

precondition[n][i][2] = value;

if (!there_already) {
precondition[n][i][0] = order_of_conditions[behaviour_n_monitoring_precond(n)][0];
precondition[n][i][1] = order_of_conditions[behaviour_n_monitoring_precond(n)][1];
precondition[n][i + 1][1] = 99;
}

clear_monitoring_stats(n);
}

-----
clear_monitoring_stats Reset the statistics for the given behaviour to zero, and reset the monitoring clock
Argument:
n = behaviour number
-----
void clear_monitoring_stats(char n)

```

```

char i, j;

for (i = 0; i < 2; i++)
  for (j = 0; j < 2; j++) {
    behav_mon_prec_pos_stats[n][i][j] = 0.0;
    behav_mon_prec_neg_stats[n][i][j] = 0.0;
  }

monitoring_clock(n) = 0;
}

/*-----
get_behaviour_number Function returns the real number of a behaviour given its group number and number within that
group.
Arguments:
group = group number
group_behaviour = number of the behaviour within that group
-----*/
char get_behaviour_number(char group, char group_behaviour)
{
  int answer = 0;

  for (; group-- > 0;)
    answer += number_of_behaviours_in_group(group);

  return (char) (answer + group_behaviour);
}

/*-----
update_output_file Write data for this time-step to the output file referred to by the 'ofp' pointer.
The template for this file is as follows:-
-----*/

time_step_counter          x 1
condition vectors:        x 1
precondition               x 4 x 13 x 3
behaviour_active          x 4
behaviour_monitoring      x 4
pos_stats                 x 4 x 2 x 2
neg_stats                 x 4 x 2 x 2
behaviour_n_monitoring_precond x 4
monitoring_clock          x 4
behav_mon_prec_pos_stats  x 4 x 2 x 2
behav_mon_prec_neg_stats  x 4 x 2 x 2
percentage positive feedback x 1
percentage negative feedback x 1
relevance                 x 4
reliability               x 4
-----*/

void update_output_file(void)
{
  int i, j, k;

  fprintf(ofp, "%-5i ", time_step_counter);
  for (i = 0; i < 3; i++)
    for (j = 0; j < 4; j++)
      fprintf(ofp, "%-11 ", (i == 0) ? leg_n_up[j] : (i == 1) ? leg_n_forward[j] : leg_n_backward[j]);
  for (i = 0; i < 4; i++)
    for (j = 0; j < 13; j++)
      for (k = 0; k < 3; k++)
        fprintf(ofp, "%-21 ", precondition[i][j][k]);
  for (i = 0; i < 4; i++)
    fprintf(ofp, "%-11 ", behaviour_active[i]);
  for (i = 0; i < 4; i++)
    fprintf(ofp, "%-11 ", behaviour_monitoring[i]);
  for (i = 0; i < 4; i++)
    for (j = 0; j < 2; j++)
      for (k = 0; k < 2; k++)
        fprintf(ofp, "%-7.3f ", pos_stats[i][j][k]);
  for (i = 0; i < 4; i++)
    for (j = 0; j < 2; j++)
      for (k = 0; k < 2; k++)
        fprintf(ofp, "%-7.3f ", neg_stats[i][j][k]);
  for (i = 0; i < 4; i++)
    fprintf(ofp, "%-21 ", behaviour_n_monitoring_precond[i]);
  for (i = 0; i < 4; i++)
    fprintf(ofp, "%-41 ", monitoring_clock[i]);
  for (i = 0; i < 4; i++)
    for (j = 0; j < 2; j++)
      for (k = 0; k < 2; k++)
        fprintf(ofp, "%-7.3f ", behav_mon_prec_pos_stats[i][j][k]);
  for (i = 0; i < 4; i++)
    for (j = 0; j < 2; j++)
      for (k = 0; k < 2; k++)
        fprintf(ofp, "%-7.3f ", behav_mon_prec_neg_stats[i][j][k]);
  fprintf(ofp, "%-31 ", get_fb_stat(pos_fb_array));
  fprintf(ofp, "%-31 ", get_fb_stat(neg_fb_array));
  for (i = 0; i < 4; i++)
    fprintf(ofp, "%-5.2f ", relevance(i));
  for (i = 0; i < 4; i++)
    fprintf(ofp, "%-4.2f ", reliability(i));
  fprintf(ofp, "\n");
}

```

```

/*
  ISSTABLE Program to check whether Smudge will be stable for a given set of leg angles.
*/
#include "top.h"

float get_normal(int, int, int);
int number_on_ground(void);
int get_sign(int, int [], float []);
void get_stability_info(float []);
void check_stable(int, float []);

/*
  get_stability_info Returns a number indicating how stable the robot is. If 3 or 4 legs are on the ground, then the
  value returned is the shortest distance between the robot CoM and the boundary of the polygon of
  support defined by the leg positions. A negative distance indicates that the robot is unstable.
  If fewer than 3 legs are on the ground, then -999.9 is returned, indicating that the robot is
  very unstable.
*/
void get_stability_info(float answer[3])
{
  int onground;
  onground = number_on_ground();
  if (onground < 3)
    answer[0] = -999.9;
  else
    check_stable(onground, answer);
  return;
}

/*
  check_stable Called by stable if there are 3 or 4 legs on the ground. Calculates whether the robot CoM lies within
  the polygon of support, and returns the shortest distance from the CoM to the boundary of the polygon.
*/
void check_stable(int onground, float answer[3])
{
  int i, j;
  int n = 0;
  int index[4];

  float normal[4][2];
  float gradient[4];
  float point[4][2];
  float distance[4];
  float min_distance;

  for (i = 0; i <= 3; i++)
    if (pos[i][2] < offset[i][2])
      index[n++] = i;

  for (i = 0; i < onground; i++)
    for (j = 0; j <= 1; j++)
      normal[i][j] = get_normal(index[i], index[(i + 1) % onground], j);

  for (i = 0; i < onground; i++)
    if (fabs(normal[i][0]) < 0.001)
      point[i][0] = 0.0;
      point[i][1] = pos[index[i]][1];
    else
      gradient[i] = normal[i][1] / normal[i][0];
      point[i][0] = ((pos[index[i]][0] * normal[i][0]) + (pos[index[(i + 1) % onground]][0] * normal[i][1]))
        / (normal[i][0] * gradient[i] + normal[i][1]);
      point[i][1] = gradient[i] * point[i][0];
      distance[i] = (float)
        sqrt(pow((double) point[i][0], 2.0) + pow((double) point[i][1], 2.0));
      if (i == 0 || distance[i] < min_distance)
        min_distance = distance[i];
        answer[1] = (float) index[i];
        answer[2] = (float) index[(i + 1) % onground];
  }

  answer[0] = (float) get_sign(onground, index, point) * min_distance;
  return;
}

/*
  get_normal Returns the nth component of the normal vector of the line connecting points i and j.
  Arguments:
    i, j, n
*/
float get_normal(int i, int j, int n)
{
  int p, sign;
  p = i - j;
  sign = (n == 0) ? 1 : -1;
  return (float) sign * (pos[i][p] - pos[j][p]);
}

/*
  number_on_ground Returns the number of feet on the ground.
*/

```

```

int number_on_ground(void)
{
    char i;
    char onground = 0;
    for (i = 0; i <= 3; i++)
        onground += (pos[i][2] < offset[i][2]); /*
    onground += (beta[i] == LEG_DOWN) ? 1 : 0;
    return onground;
}

-----
get_sign Returns 1 if robot CoM lies within polygon of support defined by feet, -1 otherwise.
-----
int get_sign(int onground, int index[4], float point[4][2])
{
    int sign; /* Remember, robot CoM is at (0,0) */
    switch(onground) {
    case 3: /* Case: 3 legs are on the ground */
        if (index[0] == 1 || index[1] == 2) /* i.e. if legs 0 or 1 are off ground */
            sign = (point[0][0] > 0 && point[1][1] < 0 && point[2][0] < 0) ? 1 : -1;
        else
            sign = (point[0][1] > 0 && point[1][0] > 0 && point[2][0] < 0) ? 1 : -1;
        break;
    case 4: /* Case: 4 legs are on the ground */
        sign = (point[0][1] > 0 && point[1][0] > 0 && point[2][1] < 0 && \
            point[3][0] < 0) ? 1 : -1;
        break;
    }
    return sign;
}

```

```

/*
FOG      Functions to return cartesian coordinates of the robot's legs (in the robot's coordinate frame) given
the leg angles.
*/

#include "top.h"
float pos[4][3];
void get_positions(void);
float pos1(char, char, float, float);
float pos2(char, char, float, float);
double radians(float);

/*
get_positions      Find (x,y,z) coordinates (in robot coordinate frame) for each foot.
*/
void get_positions(void)
{
    char i, j;
    for (i = 0; i <= 3; i++)
        for (j = 0; j <= 2; j++)
            pos[i][j] = offset[i][j] + pos1(i, j, (float) alpha[i], (float) beta[i]) + pos2(i, j, (float) alpha[i], (float) beta[i]);
}

/*
pos1      Return requested component of position of top of leg relative to servo axis of rotation.
Arguments:
    i = leg number (0 - 3)
    j = ordinate (0=x, 1=y, 2=z)
    angle1 = alpha angle for leg i
    angle2 = beta angle for leg i
*/
float pos1(char i, char j, float angle1, float angle2)
{
    char sign;
    switch (j) {
    case 0:
        sign = (i == 1 || i == 2) ? -1 : 1;
        return (float)
            (sign * SERVO1_L1 * sin(radians(angle1)));
    case 1:
        sign = (i == 2 || i == 3) ? -1 : 1;
        return (float)
            (sign * SERVO1_L1 * cos(radians(angle1)));
    case 2:
        return 0.0;
    default:
        return 999.9;
    }
}

/*
pos2      Return requested component of position of foot of leg relative to top of leg.
Arguments:
    i = leg number (0 - 3)
    j = ordinate (0=x, 1=y, 2=z)
    angle1 = alpha angle for leg i
    angle2 = beta angle for leg i
*/
float pos2(char i, char j, float angle1, float angle2)
{
    int sign;
    switch (j) {
    case 0:
        sign = (i == 0 || i == 3) ? -1 : 1;
        return (float)
            (sign * LEG_LENGTH * cos(radians(angle2)) *
            cos(radians(angle1)));
    case 1:
        sign = (i == 2 || i == 3) ? -1 : 1;
        return (float)
            (sign * LEG_LENGTH * cos(radians(angle2)) *
            sin(radians(angle1)));
    case 2:
        return (float) (- LEG_LENGTH * sin(radians(angle2)));
    default:
        return 999.9;
    }
}

/*
radians      Convert degrees into radians
Arguments:
    degrees (integer)
*/
double radians(float x)
{
    return ((double) x) / 180.0 * M_PI;
}

```

---

LENGINS.H Specifies the dimensions of Smudge, plus the beta angles for raised and lowered legs.

---

```
#define BODY_LENGTH 250
#define BODY_BREADTH 65
#define SERVO1_L1 20
#define SERVO2_L1 7
#define SERVO_POSITION 30
#define LEG_VERT_OFFSET 20
#define LEG_LENGTH 150
#define LEG_DOWN 30
#define LEG_UP -10
```

```

TOP.H Specifies Include files, standard definitions and external functions and variables.

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "useful.h"
#include "lengths.h"

#define offsetx ((float) BODY_BREADTH / 2.0) + SERVO2_L1
#define offsety ((float) BODY_LENGTH / 2.0) - SERVO_POSITION
#define offsetz (float) LEG_VERT_OFFSET
#define FB_STAT_SLICE (int) 200

extern float pos1(char, char, float, float); /* functions defined in */
extern float pos2(char, char, float, float); /* file pos.c */
extern double radians(float);

extern float get_normal(int, int, int); /* functions defined in */
extern int number_on_ground(void); /* file lstable.c */
extern int get_sign(int, int [], float []);
extern void get_stability_info(float []);
extern void check_stable(int, float []);

extern void initialise_leg_positions(void); /* functions defined in */
extern void initialise_learning(void); /* file algorithm.c */
extern void next_time_step(float [], float [], bool *);

extern int get_fb_stat(char []); /* function defined in */
/* sim.c */

extern int alpha[4], beta[4]; /* global variables */
extern float pos[4][3];
extern const float offset[4][3];
extern int advance_angle;
extern char precond_data[4][13][3], monitor_data[4][2];
extern int monitor_clock_data[4];
extern char pos_fb_array[FB_STAT_SLICE], neg_fb_array[FB_STAT_SLICE];
extern int time_step_counter;
extern FILE *ofp;

```

# Appendix C

## Program Code for the Simulator

```
-----
SIM          XView code and general control routines for the simulation
-----

#include "xview/xv_headers.hh"
#include "top.h"

#define SHRINK_FACTOR          (float) 2.0
#define S_BODY_LENGTH         (float) ((float) BODY_LENGTH / SHRINK_FACTOR)
#define S_BODY_BREADTH        (float) ((float) BODY_BREADTH / SHRINK_FACTOR)
#define S_BODY_TOLEFT_X       (float) (S_BODY_LENGTH / 2.0)
#define S_BODY_TOLEFT_Y       (float) (S_BODY_BREADTH / 2.0)
#define s_offsetx              (float) (offsetx / SHRINK_FACTOR)
#define s_offsety              (float) (offsety / SHRINK_FACTOR)
#define s_offsetz              (float) (offsetz / SHRINK_FACTOR)

const float offset[4][3] = {
    {-offsetx, offsety, -offsetz},
    { offsetx, offsety, -offsetz},
    { offsetx, -offsety, -offsetz},
    {-offsetx, -offsety, -offsetz}
};

const float s_offset[4][3] = {
    {-s_offsetx, s_offsety, -s_offsetz},
    { s_offsetx, s_offsety, -s_offsetz},
    { s_offsetx, -s_offsety, -s_offsetz},
    {-s_offsetx, -s_offsety, -s_offsetz}
};

typedef struct position {
    int      x_coord;

    int      y_coord;
    float    hip_pos[4][3];
    float    foot_pos[4][3];
} Position;

Frame      frame;
Display    *dpy;
GC         gc;
Window     canvas_win;
Panel      panel1, panel2;
Canvas     canvas;
struct timeval timer;
Panel_item slider, leg[4][2], info_msg[6], behaviour_msg[4][3], stability_msg;
Position   oldposition, newposition;
bool       last_fb = false;
char       go_ahead = FALSE;
char       info[6][10];
char       beta_message[40];
char       stability_string[20];
char       behaviour_info[4][3][43];
char       precond_data[4][13][3], monitor_data[4][2];
char       pos_fb_array[FB_STAT_SLICE], neg_fb_array[FB_STAT_SLICE];
int        monitor_clock_data[4];
int        alpha[4], beta[4];
float      total_x = 0.0, total_y = 0.0;
FILE       *ofp; /* pointer to output file */

Notify_value animate(void);
Position      make_position(int, int);
Position      get_new_position(Position, float {});
char          itoa(int);
char          *get_precondition_string(char);
char          *get_condition_string(char, char);
int           get_fb_stat(char array[]);
void          delete(Position);
void          draw(Position, int, int, float);
void          DrawDottedLine(int, int, int, int, int);
void          swap(int *, int *, int *, int *);
void          update_display_data(int);
void          update_statistics(int);
void          initialise_statistics(void);
void          change_alpha(Panel_item), change_beta(Panel_item);

main(int argc, char *argv[])
{
    void      quit(void), start(void), stop(void);
    void      adjust_speed(Panel_item, int);
    void      canvas_repaint(Canvas, Xv_Window, Display, Window, Xv_xrectlist);

    char      rows, cols, i, index;
    char      *text[] = {"Front Left", "Front Right", "Rear Right", "Rear Left"};
    char      *lines = "-----";
}
```

```

char      *label[] = {"Elapsed time-steps :", "% positive feedback :", \
                    "% negative feedback :", "Distance travelled (cm)", \
                    "    on x-axis :", "    on y-axis :"};

char      behaviour_text[4][13];
int       button_x = 4, button_y = 100;           /* origin of panel items for controlling leg ang
int       info_x = button_x, info_y = button_y - 20; /* origin of panel items for status information
int       stability_x = INFO_x + 85, stability_y = info_y + 140; /* origin of stability message

time_t    now;                                   /* used for time-stamping the output file */

strcpy(beta_message, "beta angles: down = ");
strcpy(beta_message, itoa(LEG_DOWN));
strcpy(beta_message, ", up = ");
strcpy(beta_message, itoa(LEG_UP));

for (i = 0; i < 4; i++) {                       /* initialise array of message strings to displa
strcpy(behaviour_text[i], "BEHAVIOUR ");        /* behaviour numbers
strcpy(behaviour_text[i], itoa(i));
strcpy(behaviour_text[i], ":");
}

initialise_leg_positions();

xv_init (XV_INIT_ARC_PTR_ARGV, sargc, argv, NULL);

frame = (Frame) xv_create(NULL, FRAME,
                          FRAME_LABEL,      "Squidge Simulator",
                          XV_WIDTH,        985,
                          XV_HEIGHT,      545,
                          NULL);

panell = (Panel) xv_create(frame, PANEL,
                           PANEL_LAYOUT,   PANEL_VERTICAL,
                           OPENWIN_SHOW_BORDERS, TRUE,
                           XV_WIDTH,      780,
                           XV_HEIGHT,    540,
                           XV_X,         0,
                           XV_Y,         0,
                           NULL);

xv_create(panell, PANEL_BUTTON,
          PANEL_LABEL_STRING, "Start",
          PANEL_NOTIFY_PROC, start,
          NULL);

xv_create(panell, PANEL_BUTTON,
          XV_X,               60,
          XV_Y,               4,
          PANEL_LABEL_STRING, "Stop",
          PANEL_NOTIFY_PROC, stop,
          NULL);

xv_create(panell, PANEL_BUTTON,
          XV_X,               220,
          XV_Y,               4,
          PANEL_LABEL_STRING, "Quit",
          PANEL_NOTIFY_PROC, quit,
          NULL);

slider = (Panel_Item) xv_create(panell, PANEL_SLIDER,
                                PANEL_LABEL_STRING, "Speed",
                                PANEL_SHOW_VALUE, FALSE,
                                PANEL_VALUE, 50,
                                PANEL_NOTIFY_PROC, adjust_speed,
                                NULL);

xv_create(panell, PANEL_MESSAGE,
          PANEL_LABEL_STRING, lines,
          NULL);

for (rows = 0; rows < 2; rows++)
  for (cols = 0; cols < 2; cols++) {
    index = (cols * rows) + ((1 - cols) * (3 - rows)); /* index is set to the correct leg no. as used by alpha[] etc */
    xv_create(panell, PANEL_MESSAGE,
              PANEL_LABEL_STRING, text[index],
              XV_X,               button_x + (136 * cols),
              XV_Y,               button_y + (80 * rows),
              NULL);
    xv_create(panell, PANEL_MESSAGE,
              PANEL_LABEL_STRING, "alpha",
              PANEL_LABEL_BOLD, TRUE,
              XV_X,               button_x + (136 * cols),
              XV_Y,               button_y + 20 + (80 * rows),
              NULL);
    leg[index][0] = (Panel_Item) xv_create(panell, PANEL_NUMERIC_TEXT,
                                           XV_X,               button_x + 50 + (136 * cols),
                                           XV_Y,               button_y + 20 + (80 * rows),
                                           PANEL_MAX_VALUE, 45,
                                           PANEL_MIN_VALUE, -45,
                                           PANEL_VALUE, alpha[index],
                                           PANEL_VALUE_DISPLAY_LENGTH, 3,
                                           PANEL_VALUE_STORED_LENGTH, 3,
                                           PANEL_NOTIFY_PROC, change_alpha,
                                           PANEL_CLIENT_DATA, index,
                                           NULL);
    leg[index][1] = (Panel_Item) xv_create(panell, PANEL_CHOICE,
                                           XV_X,               button_x + (136 * cols),
                                           XV_Y,               button_y + 40 + (80 * rows),
                                           PANEL_CHOICE_STRINGS, "Down", "Up", NULL,
                                           PANEL_VALUE, 0,
                                           PANEL_NOTIFY_PROC, change_beta,
                                           PANEL_CLIENT_DATA, index,
                                           NULL);
  }

xv_create(panell, PANEL_MESSAGE,
          XV_X,               button_x,
          XV_Y,               button_y - 160,
          PANEL_LABEL_STRING, beta_message,

```

```

        NULL);
    xv_create(panel1, PANEL_MESSAGE,
        PANEL_LABEL_STRING, lines,
        NULL);

    for (rows = 0; rows < 6; rows++) {
        xv_create(panel1, PANEL_MESSAGE,
            XV_X, info_x,
            XV_Y, info_y + (18 * rows),
            PANEL_LABEL_STRING, label[rows],
            NULL);
        info_msg[rows] = (Panel_item) xv_create(panel1, PANEL_MESSAGE,
            XV_X, info_x + 150,
            XV_Y, info_y + (18 * rows),
            PANEL_LABEL_STRING, info[rows],
            NULL);
    }
    xv_create(panel1, PANEL_MESSAGE,
        PANEL_LABEL_STRING, lines,
        NULL);

    stability_msg = (Panel_item) xv_create(panel1, PANEL_MESSAGE,
        XV_X, stability_x,
        XV_Y, stability_y,
        PANEL_LABEL_STRING, stability_string,
        NULL);

    panel2 = (Panel) xv_create(frame, PANEL,
        PANEL_LAYOUT, PANEL_HORIZONTAL,
        OPENWIN_SHOW_BORDERS, TRUE,
        XV_WIDTH, 700,
        XV_HEIGHT, 140,
        XV_X, 280,
        XV_Y, 0,
        NULL);

    for (rows = 0; rows < 2; rows++)
        for (cols = 0; cols < 2; cols++) {
            index = (rows * 2) + cols;
            xv_create(panel2, PANEL_MESSAGE,
                PANEL_LABEL_STRING, behaviour_text[index],
                PANEL_LABEL_BOLD, TRUE,
                XV_X, 4 + (350 * cols),
                XV_Y, 4 + (70 * rows),
                NULL);
            xv_create(panel2, PANEL_MESSAGE,
                PANEL_LABEL_STRING, "Preconditions",
                XV_X, 4 + (350 * cols),
                XV_Y, 24 + (70 * rows),
                NULL);
            xv_create(panel2, PANEL_MESSAGE,
                PANEL_LABEL_STRING, "-----",
                XV_X, 4 + (350 * cols),
                XV_Y, 34 + (70 * rows),
                NULL);
            for (i = 0; i < 3; i++)
                behaviour_msg[index][i] = (Panel_item) xv_create(panel2, PANEL_MESSAGE,
                    PANEL_LABEL_STRING, behaviour_info[index][i],
                    XV_X, 4 + ((i == 0) ? 0 : (i == 1) ? 155 : 215) + (350 * cols),
                    XV_Y, 44 + (70 * rows),
                    NULL);
        }

    canvas = (Canvas) xv_create(frame, CANVAS,
        CANVAS_REPAINT_PROC, canvas_repaint,
        CANVAS_X_PAINT_WINDOW, TRUE,
        OPENWIN_SHOW_BORDERS, TRUE,
        XV_WIDTH, 700,
        XV_HEIGHT, 400,
        XV_X, 282,
        XV_Y, 142,
        NULL);
    canvas_win = (Window) xv_get(canvas_paint_window(canvas), XV_XID);

    dpy = (Display *) xv_get(frame, XV_DISPLAY);
    gc = DefaultGC(dpy, DefaultScreen(dpy));

    oldposition = make_position(200, 0);
    initialise_statistics();
    initialise_learning();

    ofp = fopen("data/data.out", "a");
    now = time(NULL);
    fprintf(ofp, "SMUDGE Simulation: Session began %s\n", ctime(&now));
    srand(time(NULL));
    xv_main_loop(frame);
    fclose(ofp);
    exit(0);

}

void quit(void) /* called by Quit button */
{
    xv_destroy_safe(frame);
}

void start(void) /* called by Start button */
{
    int value;

    gc_shed = TRUE;
    value = (int) xv_get/slider, PANEL_VALUE);
    if (value > 0) {

```

```

timer.it_value.tv_usec = (130 - value) * 1000;
timer.it_interval.tv_usec = (130 - value) * 1000;
notify_set_itimer_func(frame, animate, ITIMER_REAL, &timer, NULL);
}
else
/* turn it off */
notify_set_itimer_func(frame, NOTIFY_FUNC_NULL, ITIMER_REAL, NULL, NULL);

/* advance one frame */
animate();
}

void stop(void) /* called by Stop button */
{
go_ahead = FALSE;
notify_set_itimer_func(frame, NOTIFY_FUNC_NULL, ITIMER_REAL, NULL, NULL);
}

void canvas_repaint(Canvas canvas, Xv_Window paint_window, Display *dpy, Window xwin, Xv_xrectlist *area)
{
int width, height;

width = (int) xv_get(paint_window, XV_WIDTH);
height = (int) xv_get(paint_window, XV_HEIGHT);
XClearArea(dpy, xwin, 0, 0, width, height, FALSE);
drawoldposition, 0, 0, 0.0);
}

Notify_value animate(void) /* draw Smudge at next */
/* time-step */
{
float advance_values[2], stable_values[3];

next_time_step(advance_values, stable_values, &last_fb);
newposition = get_new_position(oldposition, advance_values);
delete(oldposition);
draw(newposition, (int) stable_values[1], (int) stable_values[2], stable_values[0]);
oldposition = newposition;

return NOTIFY_DONE;
}

void adjust_speed(Panel_item item, int value)
{
if (go_ahead == TRUE) {
if (value > 0) {
timer.it_value.tv_usec = (130 - value) * 1000;
timer.it_interval.tv_usec = (130 - value) * 1000;
notify_set_itimer_func(frame, animate, ITIMER_REAL, &timer, NULL);
} else
/* turn it off */
notify_set_itimer_func(frame, NOTIFY_FUNC_NULL, ITIMER_REAL, NULL, NULL);
}
}

void change_alpha(Panel_item item) /* manual adjustment of */
/* alpha values from */
/* control panel */
{
int i = (int) xv_get(item, PANEL_CLIENT_DATA);
alpha[i] = xv_get(item, PANEL_VALUE);
}

void change_beta(Panel_item item) /* manual adjustment of */
/* beta values from */
/* control panel */
{
int i = (int) xv_get(item, PANEL_CLIENT_DATA);
beta[i] = (xv_get(item, PANEL_VALUE) == 1) ? LEG_DOWN : LEG_UP;
}

Position make_position(int x, int y) /* create a new Position */
/* data structure */
{
Position temp;
char i, j;
int half_height = ((int) xv_get(canvas, CANVAS_HEIGHT)) / 2;

get_positions();

temp.x_coord = x;
temp.y_coord = half_height - y * half_height;
for (i = 0; i < 4; i++)
for (j = 0; j < 3; j++) {
temp.hip_pos[i][j] = s_offset[i][j] - (pos[i][j], ((float) alpha[i], (float) beta[i]) / SHRINK_FACTOR);
temp.foot_pos[i][j] = pos[i][j] / SHRINK_FACTOR;
}

return temp;
}

Position get_new_position(Position old_pos, float advance_values[]) /* return a new Position data structure */
/* based on the old structure and advance */
/* values passed into the function */
{
Position temp;
int half_height = ((int) xv_get(canvas, CANVAS_HEIGHT)) / 2;

temp.x_coord = (old_pos.x_coord - (int) S_BODY_TOLEFT_X - (int) (advance_values[1] / SHRINK_FACTOR)) \
% ((int) xv_get(canvas, CANVAS_WIDTH)) - (int) S_BODY_LENGTH;
temp.y_coord = half_height - (old_pos.y_coord - half_height - (int) (advance_values[0] / SHRINK_FACTOR)) * half_height;

for (i = 0; i < 4; i++)
for (j = 0; j < 3; j++) {
temp.hip_pos[i][j] = s_offset[i][j] - (pos[i][j], ((float) alpha[i], (float) beta[i]) / SHRINK_FACTOR);
temp.foot_pos[i][j] = pos[i][j] / SHRINK_FACTOR;
}
}

```

```

        NULL);
    }
    for (i = 0; i < 6; i++)
        if (strcmp(char *) xv_get(info_mag[i], PANEL_LABEL_STRING), info[i] != 0)
            xv_set(info_mag[i],
                PANEL_LABEL_STRING, info[i],
                NULL);
    if (strcmp(char *) xv_get(stability_mag, PANEL_LABEL_STRING), stability_string, 12) != 0)
        xv_set(stability_mag,
            PANEL_LABEL_STRING, stability_string,
            NULL);
    for (i = 0; i < 4; i++)
        for (j = 0; j < 3; j++)
            if (strcmp(char *) xv_get(behaviour_mag[i][j], PANEL_LABEL_STRING), behaviour_info[i][j] != 0)
                xv_set(behaviour_mag[i][j],
                    PANEL_LABEL_STRING, behaviour_info[i][j],
                    NULL);
}

void update_statistics(int stable_flag)
{
    char i;
    strcpy(info[0], ltoa(time_step_counter));
    strcpy(info[1], ltoa(get_fb_stat(pos_fb_array)));
    strcpy(info[2], ltoa(get_fb_stat(neg_fb_array)));
    strcpy(info[4], ltoa((int) (total_x / 10.0)));
    strcpy(info[5], ltoa((int) (total_y / 10.0)));
    if (stable_flag)
        strcpy(stability_string, "Smudge is stable");
    else
        strcpy(stability_string, "SMUDGE IS UNSTABLE!");
    for (i = 0; i < 4; i++) {
        strcpy(behaviour_info[i][0], get_precondition_string(i));
        if (monitor_clock_data[i] == 0)
            strcpy(behaviour_info[i][1], "");
        else
            strcpy(behaviour_info[i][1], get_condition_string(monitor_data[i][0], monitor_data[i][1]));
        strcpy(behaviour_info[i][2], ltoa(monitor_clock_data[i]));
    }
}

char *get_precondition_string(char n)
/* function returns a char array of all */
/* current preconditions for behaviour */
/* n suitable for display on the panel */
{
    char i;
    char answer[43];
    strcpy(answer, "");
    for (i = 0; precond_data[n][i][1] != 99; i++) {
        if (i > 0)
            strcat(answer, ", ");
        strcat(answer, get_condition_string(precond_data[n][i][0], precond_data[n][i][1]));
        strcat(answer, " ");
        strcat(answer, (precond_data[n][i][2] == true) ? "T" : "F");
    }
    return answer;
}

char *get_condition_string(char n, char m)
/* function returns a char array */
/* describing a particular condition */
/* (e.g. LFI suitable for display on */
/* the panel */
/* 99 is a default value */
{
    char answer[5];
    if (n == 99 || m == 99)
        strcpy(answer, "");
    else {
        strcpy(answer, "L");
        strcat(answer, ltoa(m));
        strcat(answer, (n == 0) ? "U" : (n == 1) ? "F" : "S");
    }
    return answer;
}

int get_fb_stat(char array[])
{
    int i, counter = 0;
    if (time_step_counter == 0)
        return 0;
    for (i = 0; i < FB_STAT_SLICE && i < time_step_counter; i++)
        counter += array[i];
    return (counter * 100 / i);
}

void initialize_statistics(void)
/* Called once, at the start of the */
/* simulation */
{
    char i, j;
    strcpy(info[0], "0");
    strcpy(info[1], "0");
    strcpy(info[2], "0");
    strcpy(info[3], "");
    strcpy(info[4], "0");
}

```

```

strcpy(info[5], "0");
for (i = 0; i < 4; i++) {
  for (j = 0; j < 2; j++) {
    precond_data[i][0][j] = (j == 1) ? 99 : 0;
    monitor_data[i][j] = 99;
  }
  monitor_clock_data[i] = 0;
  for (j = 0; j < 3; j++)
    strcpy(behaviour_info[i][j], "");
}
}

void swap(int *x, int *y_1, int *x_2, int *y_2)
{
  int temp_x, temp_y;

  temp_x = *x_1;
  temp_y = *y_1;
  *x_1 = *x_2;
  *y_1 = *y_2;
  *x_2 = temp_x;
  *y_2 = temp_y;
}

char *itoa(int n) /* Converts an integer to a char array */
{
  int i;
  int negative = FALSE; /* length of returned string is at least 2 */
  char count = 2; /* (including \0 at end) */
  char string[10]; /* count number of digits */

  for (i = n; (i /= 10) != 0; count++); /* to make space for the minus sign */

  if (n < 0) {
    count++;
    n = -n;
    negative = TRUE;
  }

  for (string[--count] = '\0'; count > 0; ) {
    string[--count] = (count == 0 && negative) ? '-' : n % 10 + '0';
    n /= 10;
  }

  return string;
}

```

# Appendix D

## Program Code for the Microcontrollers

```
/ STAGE 1 *****
/ Output values onto Port B pin 1. Cycle through a series of three
/ pulse widths to drive the servo through a cycle of three positions.
/
/ At 4MHz, instruction cycles take 0.501 us. The RTCC prescaler is
/ off, and the RTCC starts at 256 - (32-2) = 226, giving an interrupt
/ every 0.032ms.
/
/ Pin Outs
/ Bit      Port B
/ 0        unused
/ 1        output to servo
/ 2-7      unused

include "c:\picasm\picreg.equ"

OPTIONS EQU      81h
start1 EQU      0Ch          ; duration of hi signal
count1 EQU      0Dh          ; 1ms timer
signal EQU      0Eh          ; 0 = low, 1 = high
ct20a EQU      0Fh          ; 20ms timer, LSB
ct20b EQU      10h          ; 20ms timer, MSB
sending EQU     11h          ; signal high flag
count_m EQU     12h
count_n EQU     13h
count_l EQU     14h

pos0 EQU      D'31'          ; pos0=(31x.032)=0.99ms
pos1 EQU      D'47'          ; pos1=(47x.032)=1.50ms
pos2 EQU      D'63'          ; pos2=(63x.032)=2.02ms
strt20a EQU   D'113'         ; signal every
strt20b EQU   D'3'          ; (2*256-113)x.032=20ms

rtstrt EQU    D'226'         ; for a .032ms interrupt
                          ; (226 = 256 - (32 - 2))

ORG      00h
GOTO    init                ; initialisation

ORG      04h
BCF     INTCON, GIE         ; interrupt routine
MOVLM   rtstrt             ; global interrupt disable
MOVWF   RTCC               ; reset RTCC counter
BCF     INTCON, RTIF       ; clear RTCC flag
DECFSE ct20a, 1           ; update 20ms timer
GOTO    update1            ; if 20ms not expired
DECFSE ct20b, 1
GOTO    set20ct            ; reset counters
MOVLM   strt20a            ; reset counters
MOVWF   ct20a              ; MOVLM   strt20b
MOVWF   ct20b
MOVWF   start1, w
MOVWF   count1
BSF     sending, 0         ; set sending flag
CALL    sendhi
BSF     INTCON, GIE       ; global interrupt enable
RETFIE

update1 BTFSC   sending, 0   ; is signal high?
CALL   count1
BSF     INTCON, GIE       ; global interrupt enable
RETFIE

set20ct MOVLM   h'FF'
MOVWF   ct20a
GOTO    update1

count1 DECFSE   count1, 1    ; update 1ms timer
RETURN
BCF     sending, 0         ; clear sending flag
CALL    sendlo
RETURN

sendhi MOVLM   1h
MOVWF   signal
CALL    send
RETURN

sendlo CLRF    signal
CALL    send
RETURN

send   MOVLM   TRISB
MOVWF   FSR              ; indirect to TRISB register
MOVLM   b'00000000'      ; set all Port B pins to o/p
```

```

RETURN
send2 BCF Port_B, 1
RETURN

A2D BSF Port_B, 0 ; for debugging
CLRF ADCON0 ; disable A/D conversions
BCF INTCON, ADIE ; disable A/D interrupts
MOVLW TRISA
MOVWF FSR ; indirect to TRISA register
MOVLW b'00000000' ; set all Port B pins to o/p
MOVWF 0h
MOVLW b'00000011' ; clear bits 2-7 of Port B
ANDWF Port_B, 1 ; read A/D conversion result
MOVWF ADRES, w
MOVWF result
ANDLW b'11111100' ; mask out bits 0 and 1
IORWF Port_B, 1 ; write to bits 2-7 of Port B
CALL config ; reconfigure Port A in case
; of noise interference
BCF Port_B, 0 ; for debugging
RETFIE

ORG 50h
init CLRF sending ; clear sending flag
MOVLW OPTIONS
MOVWF FSR
MOVLW b'01001000' ; set prescaler off (1:1)
MOVWF 0h
MOVLW strt20a ; initialise 20ms timer
MOVWF ct20a ;
MOVLW strt20b ;
MOVWF ct20b ;

CALL config ; Configure Port A correctly
CLRF ADCON0 ; disable A/D conversions
MOVLW b'10100000' ; Enable RTCC interrupts
MOVWF INTCON

MOVLW rtatct
MOVWF RTCC
MOVLW pos0
MOVWF count1 ; initialize ina count
BSF sending, 0
CALL sendhi ; send initial hi signal

loop MOVLW pos0 ; loop changes value of
MOVWF start1 ; start1 every 1.5 seconds
CALL wait
MOVLW pos1
MOVWF start1
CALL wait
MOVLW pos2
MOVWF start1
CALL wait
GOTO loop

wait MOVLW D'10'
MOVWF count_1
BSF ADCON0, ADON ; enable A/D convertor
BSF INTCON, ADIE ; enable A/D interrupts
BSF ADCON0, GO ; begin an A/D conversion
CALL time_1 ; wait for ~1 second
RETURN

time_s ; each NOP is 1 cycle
NOP
NOP ; running at 4MHz,
NOP ; each cycle takes .001ms
NOP
NOP
NOP ; there are 7 NOPs
DECFSZ count_s, 1 ; this takes 1 or 2 cycles
GOTO time_s ; this takes 2 cycles
RETURN ; result = wait count_s * 10 cycles
; (i.e. count_s * .01ms (~.002ms))

time_m MOVLW D'99'
MOVWF count_m
CALL time_s ; wait for .992ms
NOP
DECFSZ count_m, 1 ; the time_m loop takes 8 cycles
GOTO time_m ; (i.e. .008ms at 4MHz)
RETURN ; result = wait count_m * 1ms

time_l MOVLW D'100'
MOVWF count_l
CALL time_m ; wait for 0.1s
DECFSZ count_l, 1
GOTO time_l
RETURN ; result = wait count_l * .1s

config MOVLW TRISA
MOVWF FSR ; All Port A pins are i/p
MOVLW b'00011111'
MOVWF 0h
MOVLW ADCON1
MOVWF FSR ; All Port A pins analogue
MOVLW b'00000000'
MOVWF 0h
RETURN

```

END