ELE4307

Assignment 1

Control of Pick and Place Machine for SMT Assembly

Student Name: Kate Bowater

Student Number: U1019160

# Introduction

This report details the design of a program written in C to control a pick and place machine for assembly of Surface Mount Technology (SMT) based Printed Circuit Boards (PCBs). The machine is designed for both manual and automatic modes and is fully tested for functionality using a simulator by a POSIX compliant program.

# System Design

A Mealy state based diagram for Part A (manual) operation

A Mealy state based diagram for Part B (autonomous) operation

## Design Choices

A brief explanation of significant design choices for Part A (manual) AND Part B (autonomous) (maximum 2 pages)

### Part A

### Part B

# Testing Results

The test cases you executed for Part A (manual) AND Part B (autonomous) and the associated results. Test cases and the results are best represented in tabular form.

## Part A

## Part B

# Appendix

/\*

\*

\* pnpControl.c - the controller for the pick and place machine in manual and autonomous mode

\*

\* Platform: Any POSIX compliant platform

\* Intended for and tested on: Cygwin 64 bit

\*

\*/

#include "pnpControl.h"

// state names and numbers

#define HOME 0

#define MOVE\_TO\_FEEDER 1

#define WAIT\_1 2

#define LOWER\_CNTR\_NOZZLE 3 //lowering the centre nozzle

#define VAC\_CNTR\_NOZZLE 4 //applying the vacuum for the centre nozzle

#define RAISE\_CNTR\_NOZZLE 5 //raising the centre nozzle

#define MOVE\_TO\_CAMERA 6

#define LOOK\_UP\_PHOTO 7

#define MOVE\_TO\_PCB 8

#define CHECK\_ERROR 9

#define holdingpart 1

#define not\_holdingpart 0

/\* state\_names of up to 19 characters (the 20th character is a null terminator), only required for display purposes \*/

const char state\_name[10][20] = {"HOME ",

"MOVE TO FEEDER ",

"WAIT 1 ",

"LOWER CNTR NOZZLE ",

"VAC CNTR NOZZLE ",

"RAISE CNTR NOZZLE ",

"MOVE TO CAMERA ",

"LOOK UP PHOTO ",

"MOVE TO PCB ",

"CHECK ERROR "};

const double TAPE\_FEEDER\_X[NUMBER\_OF\_FEEDERS] = {FDR\_0\_X, FDR\_1\_X, FDR\_2\_X, FDR\_3\_X, FDR\_4\_X, FDR\_5\_X, FDR\_6\_X, FDR\_7\_X, FDR\_8\_X, FDR\_9\_X};

const double TAPE\_FEEDER\_Y[NUMBER\_OF\_FEEDERS] = {FDR\_0\_Y, FDR\_1\_Y, FDR\_2\_Y, FDR\_3\_Y, FDR\_4\_Y, FDR\_5\_Y, FDR\_6\_Y, FDR\_7\_Y, FDR\_8\_Y, FDR\_9\_Y};

const char nozzle\_name[3][10] = {"left", "centre", "right"};

int main()

{

pnpOpen();

int operation\_mode, number\_of\_components\_to\_place, res;

PlacementInfo pi[MAX\_NUMBER\_OF\_COMPONENTS\_TO\_PLACE];

/\*

\* read the centroid file to obtain the operation mode, number of components to place

\* and the placement information for those components

\*/

res = getCentroidFileContents(&operation\_mode, &number\_of\_components\_to\_place, pi);

if (res != CENTROID\_FILE\_PRESENT\_AND\_READ)

{ //throw an error if the centroid file is unreadable or not present

printf("Problem with centroid file, error code %d, press any key to continue\n", res);

getchar();

exit(res);

}

/\* state machine code for manual control mode \*/

if (operation\_mode == MANUAL\_CONTROL)

{

/\* initialization of variables and controller window \*/

int state = HOME, finished = FALSE;

char NozzleStatus = not\_holdingpart; //initiating nozzle status

char c;

int part\_counter = 0;

printf("Time: %7.2f Initial state: %.15s Operating in manual control mode, there are %d parts to place\n\n", getSimulationTime(), state\_name[HOME], number\_of\_components\_to\_place);

/\* print details of part 0 \*/

printf("Part 0 details:\nDesignation: %s\nFootprint: %s\nValue: %.2f\nx: %.2f\ny: %.2f\ntheta: %.2f\nFeeder: %d\n\n",

pi[0].component\_designation, pi[0].component\_footprint, pi[0].component\_value, pi[0].x\_target, pi[0].y\_target, pi[0].theta\_target, pi[0].feeder);

/\* loop until user quits \*/

while(!isPnPSimulationQuitFlagOn())

{

c = getKey(); //saves the value of the key pressed by the user

switch (state)

{

case HOME:

//gantry in home position, waiting for input by user to initiate movement to feeder

if (finished == FALSE && (c == '0' || c == '1' || c == '2' || c == '3' || c == '4' || c == '5' || c == '6' || c == '7' || c == '8' || c == '9'))

{

/\* the expression (c - '0') obtains the integer value of the number key pressed \*/

setTargetPos(TAPE\_FEEDER\_X[c - '0'], TAPE\_FEEDER\_Y[c - '0']);

state = MOVE\_TO\_FEEDER;

printf("Time: %7.2f New state: %.20s Issued instruction to move to tape feeder %c\n", getSimulationTime(), state\_name[state], c);

}

break;

case MOVE\_TO\_FEEDER:

//waiting for the simulator to complete movement of the gantry

if (isSimulatorReadyForNextInstruction())

{

state = WAIT\_1;

printf("Time: %7.2f New state: %.20s Arrived at feeder, waiting for next instruction\n", getSimulationTime(), state\_name[state]);

}

break;

case WAIT\_1: //waiting for next key press

//'p' for pickup

if((c == 'p') && (NozzleStatus == not\_holdingpart)) //the nozzle needs to be empty before initiating pickup

{

lowerNozzle(CENTRE\_NOZZLE);

state = LOWER\_CNTR\_NOZZLE;

printf("Time: %7.2f New state: %.20s Issued instruction to pick up part\n", getSimulationTime(), state\_name[state]);

}

//'c' for camera

if((c == 'c') && (NozzleStatus == holdingpart))

{

setTargetPos(LOOKUP\_CAMERA\_X,LOOKUP\_CAMERA\_Y); //the gantry will move to the position above the camera

state = MOVE\_TO\_CAMERA; //after the nozzle picked up a part, send the gantry to the lookup camera

printf("Time: %7.2f New state: %.20s Issued instruction to move to look-up camera\n", getSimulationTime(), state\_name[state]);

}

break;

case LOWER\_CNTR\_NOZZLE:

if (isSimulatorReadyForNextInstruction())

{

applyVacuum(CENTRE\_NOZZLE);

state = VAC\_CNTR\_NOZZLE;

printf("Time: %7.2f New state: %.20s Applying vacuum\n", getSimulationTime(), state\_name[state]);

}

break;

case VAC\_CNTR\_NOZZLE:

if (isSimulatorReadyForNextInstruction())

{

raiseNozzle(CENTRE\_NOZZLE);

state = RAISE\_CNTR\_NOZZLE;

printf("Time: %7.2f New state: %.20s Raising nozzle.\n", getSimulationTime(), state\_name[state]);

}

break;

case RAISE\_CNTR\_NOZZLE:

if (isSimulatorReadyForNextInstruction())

{

state = WAIT\_1;

NozzleStatus = holdingpart;

printf("Time: %7.2f New state: %.20s Part acquired. Ready for next instruction\n", getSimulationTime(), state\_name[state]);

}

break;

case MOVE\_TO\_CAMERA:

if (isSimulatorReadyForNextInstruction())

{

takePhoto(PHOTO\_LOOKUP);

state = LOOK\_UP\_PHOTO;

printf("Time: %7.2f New state: %.20s Taking look-up photo of part\n", getSimulationTime(), state\_name[state]);

}

break;

case LOOK\_UP\_PHOTO:

if (isSimulatorReadyForNextInstruction())

{

setTargetPos(pi[part\_counter].x\_target, pi[part\_counter].y\_target);

state = MOVE\_TO\_PCB;

printf("Time: %7.2f New state: %.20s Look-up photo taken of acquired part. Moving to PCB\n", getSimulationTime(), state\_name[state]);

}

break;

case MOVE\_TO\_PCB:

if (isSimulatorReadyForNextInstruction())

{

takePhoto(PHOTO\_LOOKDOWN);

state = CHECK\_ERROR;

printf("Time: %7.2f New state: %.20s Look-down photo taken of acquire part. Calculating misalignment errors\n", getSimulationTime(), state\_name[state]);

}

break;

case CHECK\_ERROR:

if (isSimulatorReadyForNextInstruction())

{

double errortheta = getPickErrorTheta(CENTRE\_NOZZLE);

double requested\_theta = pi[part\_counter].theta\_target - errortheta; //checking for misalignment after the part has been picked up

state = WAIT\_1;

printf("Time: %7.2f New state: %.20s the program is paused after getting requested theta %7.2f\n", getSimulationTime(), state\_name[state], requested\_theta);

printf("requested theta:%7.2f target theta:%7.2f error theta%7.2f\n", requested\_theta, pi[part\_counter].theta\_target, errortheta);

}

break;

//

// case PART\_DROP:

//

// lowerNozzle(CENTRE\_NOZZLE);

// releaseVacuum(CENTRE\_NOZZLE);

// raiseNozzle(CENTRE\_NOZZLE);

//

// break;

}

sleepMilliseconds((long) 1000 / POLL\_LOOP\_RATE);

}

}

/\* state machine code for autonomous control mode \*/

else

{

// if (operation\_mode == MANUAL\_CONTROL)

// {

// /\* initialization of variables and controller window \*/

// int state = HOME, finished = FALSE;

//

// char c;

// printf("Time: %7.2f Initial state: %.15s Operating in automatic mode, there are %d parts to place\n\n", getSimulationTime(), state\_name[HOME], number\_of\_components\_to\_place);

// /\* print details of part 0 \*/

// printf("Part 0 details:\nDesignation: %s\nFootprint: %s\nValue: %.2f\nx: %.2f\ny: %.2f\ntheta: %.2f\nFeeder: %d\n\n",

// pi[0].component\_designation, pi[0].component\_footprint, pi[0].component\_value, pi[0].x\_target, pi[0].y\_target, pi[0].theta\_target, pi[0].feeder);

//

// /\* loop until user quits \*/

// while(!isPnPSimulationQuitFlagOn())

// {

//

// c = getKey();

//

// switch (state)

// {

//

// }

// }

// }

}

pnpClose();

return 0;

}