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

It is assumed that the user will know the controls of the system and thus does not need reminding via the prompts how to initiate pickup of parts, the camera, etc.

It is assumed that all parts need to be placed in the order as sequenced in the centroid file, so an error will be presented to the user if they do not press the correct button for the feeder of the required part.

### 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

Debugging was performed throughout development of the program. Once it was determined that the program could run through a centroid file without errors, comprehensive testing was achieved by checking against Table \_.

Table \_: Testing checklist for manual mode

|  |  |  |
| --- | --- | --- |
| **Item** | **Result** | **Notes** |
|  |  |  |
|  |  |  |

## 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 LOOK\_DOWN\_PHOTO 9

#define CHECK\_ERROR 10

#define CORRECT\_ERRORS 11

#define MOVE\_TO\_HOME 12

#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[13][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 ",

"LOOK DOWN PHOTO ",

"CHECK ERROR ",

"CORRECT ERRORS ",

"MOVE TO HOME "};

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, part\_counter = 0;

char c, part\_placed = FALSE, NozzleStatus = not\_holdingpart;

double requested\_theta = 0; //the required angle theta of the nozzle position

double preplace\_diff\_x = 0, preplace\_diff\_y = 0; //difference in required gantry position and actual gantry position for preplacement

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

part\_placed = FALSE; //variable to change state actions based on whether a part has just been placed or not. Also resets upon returning to home after placing part

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

{

//check if user inputs a feeder number that is not next in the centroid file to prevent accidentally picking up wrong part

if ((c - '0') != pi[part\_counter].feeder)

{

printf("Time: %7.2f INPUT ERROR The next part is in feeder %d.\n", getSimulationTime(), pi[part\_counter].feeder);

}

else

{ //if the input is accepted, the gantry will move to the feeder position

/\* 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)) //checking if the nozzle is empty

{

lowerNozzle(CENTRE\_NOZZLE);

state = LOWER\_CNTR\_NOZZLE;

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

}

//'p' to place the part that the nozzle is currently holding

//consider applying another condition to stop user if errors are not yet corrected

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

{

lowerNozzle(CENTRE\_NOZZLE);

state = LOWER\_CNTR\_NOZZLE;

printf("Time: %7.2f New state: %.20s Issued instruction to place part on PCB. Lowering nozzle\n", getSimulationTime(), state\_name[state]);

}

//'c' for camera, should only go to the camera if the nozzle is holding a part

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]);

}

//'r' for rotate to fix the nozzle misalignment error

if(c == 'r')

{

rotateNozzle(CENTRE\_NOZZLE, requested\_theta); //rotate the nozzle by the required calculated angle theta

state = CORRECT\_ERRORS;

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

}

//'a' for adjusting the position of the gantry for preplace misalignment error

if(c == 'a')

{

amendPos(preplace\_diff\_x, preplace\_diff\_y); //corrects the position by the calculated difference x and y

state = CORRECT\_ERRORS;

printf("Time: %7.2f New state: %.20s Correcting preplace misalignment of gantry\n", getSimulationTime(), state\_name[state]);

}

if(c == 'h')

{

setTargetPos(HOME\_X,HOME\_Y);

state = MOVE\_TO\_HOME;

printf("Time: %7.2f New state: %.20s Moving to home position\n", getSimulationTime(), state\_name[state]);

}

break;

case LOWER\_CNTR\_NOZZLE:

//Need to wait until simulator is ready before moving on to vacuum

if (isSimulatorReadyForNextInstruction())

{

if(NozzleStatus == not\_holdingpart)

{ //vacuum will apply when the nozzle is empty

applyVacuum(CENTRE\_NOZZLE);

state = VAC\_CNTR\_NOZZLE;

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

}

if(NozzleStatus == holdingpart)

{ //vacuum will release the part when the nozzle is holding something

releaseVacuum(CENTRE\_NOZZLE);

part\_placed = TRUE; //counter to indicate the part has been placed

state = VAC\_CNTR\_NOZZLE;

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

}

}

break;

case VAC\_CNTR\_NOZZLE:

//wait until the vacuum action is finished before raising the 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:

//once nozzle is raised, if a part hasn't just been placed, then it is determined that a part has just been picked up

if (isSimulatorReadyForNextInstruction())

{

if (part\_placed==FALSE)

{

NozzleStatus = holdingpart;

state = WAIT\_1;

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

}

//if the vacuum has just released a part, then the part has been placed and the nozzle is free again

if (part\_placed==TRUE)

{

NozzleStatus = not\_holdingpart;

part\_counter++; //increment counter to keep track of the part number in the centroid file that has been placed

if (part\_counter != number\_of\_components\_to\_place)

{ //since there are still components to be placed, go back to Home to cycle again. Display the next set of part details

state = HOME;

printf("Time: %7.2f New state: %.20s Part %d placed on PCB successfully\n\n", getSimulationTime(), state\_name[state], (part\_counter-1));

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

pi[part\_counter].component\_designation, pi[part\_counter].component\_footprint, pi[part\_counter].component\_value, pi[part\_counter].x\_target,

pi[part\_counter].y\_target, pi[part\_counter].theta\_target, pi[part\_counter].feeder);

}

else if(part\_counter == number\_of\_components\_to\_place)

{

setTargetPos(HOME\_X,HOME\_Y);

state = MOVE\_TO\_HOME;

printf("Time: %7.2f New state: %.20s All parts have been placed! Moving to home\n", getSimulationTime(), state\_name[state]);

}

}

}

break;

case MOVE\_TO\_CAMERA:

//waiting for the gantry to move to the camera position before taking look-up photo

if (isSimulatorReadyForNextInstruction())

{

takePhoto(PHOTO\_LOOKUP);

state = LOOK\_UP\_PHOTO;

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

}

break;

case LOOK\_UP\_PHOTO:

if (isSimulatorReadyForNextInstruction())

{ //once look-up photo is taken, move the gantry to the PCB for part placement

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 acquired. Moving to PCB\n", getSimulationTime(), state\_name[state]);

}

break;

case MOVE\_TO\_PCB:

//once the gantry has finished moving to the PCB, then it is ready to take a look-down photo

if (isSimulatorReadyForNextInstruction())

{

state = LOOK\_DOWN\_PHOTO;

printf("Time: %7.2f New state: %.20s Now at PCB. Taking look-down photo\n", getSimulationTime(), state\_name[state]);

}

break;

case LOOK\_DOWN\_PHOTO:

//take the look-down photo, then move on to check for errors

takePhoto(PHOTO\_LOOKDOWN);

state = CHECK\_ERROR;

printf("Time: %7.2f New state: %.20s Look-down photo acquired. Checking for errors in alignment\n", getSimulationTime(), state\_name[state]);

break;

case CHECK\_ERROR:

//wait until the look-down photo is taken, then calculate errors

if (isSimulatorReadyForNextInstruction())

{

double errortheta = getPickErrorTheta(CENTRE\_NOZZLE); //acquire the part misalignment from the look-up photo

requested\_theta = pi[part\_counter].theta\_target - errortheta; //calculate misalignment of the part on the nozzle

preplace\_diff\_x = pi[part\_counter].x\_target - (pi[part\_counter].x\_target+getPreplaceErrorX()); //calculate the difference between the required x position and the actual x position of the gantry

preplace\_diff\_y = pi[part\_counter].y\_target - (pi[part\_counter].y\_target+getPreplaceErrorY()); //calculate the difference between the required y position and the actual y position of the gantry

state = WAIT\_1; //display the errors to the user so they are aware and then wait for instruction

printf("Time: %7.2f %19s Part misalignment error: %3.2f, preplace misalignment error: x=%3.2f y=%3.2f\n", getSimulationTime()," ", errortheta, getPreplaceErrorX(), getPreplaceErrorY());

printf("Time: %7.2f New state: %.20s Waiting for next instruction. Recommend error correction\n", getSimulationTime(),state\_name[state]);

}

break;

case CORRECT\_ERRORS:

if (isSimulatorReadyForNextInstruction())

{ //once the nozzle or gantry position has been corrected, go back to wait for next instruction

state = WAIT\_1;

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

}

break;

case MOVE\_TO\_HOME:

if (isSimulatorReadyForNextInstruction())

{

state = HOME;

printf("Time: %7.2f New state: %.20s Gantry in Home position. Provide next instruction or q to quit.\n", getSimulationTime(), state\_name[state]);

}

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;

}