Boston University

EC463 Senior Design Project

First Prototype Testing Report

NanoPack
A PACKING MACHINE FOR
NANOVIEW BIOSCIENCES

BY:

Devin Bidstrup *dbids at bu dot edu*

Joseph Walsh *jwalsh15 at bu dot edu*

Paul Stephen Hutchinson Maltaghati *psmalta at bu dot edu*

Justin Melville *jem2000 bu dot edu*

George Kent-Scheller *georgeks at bu dot edu*

Required Materials:

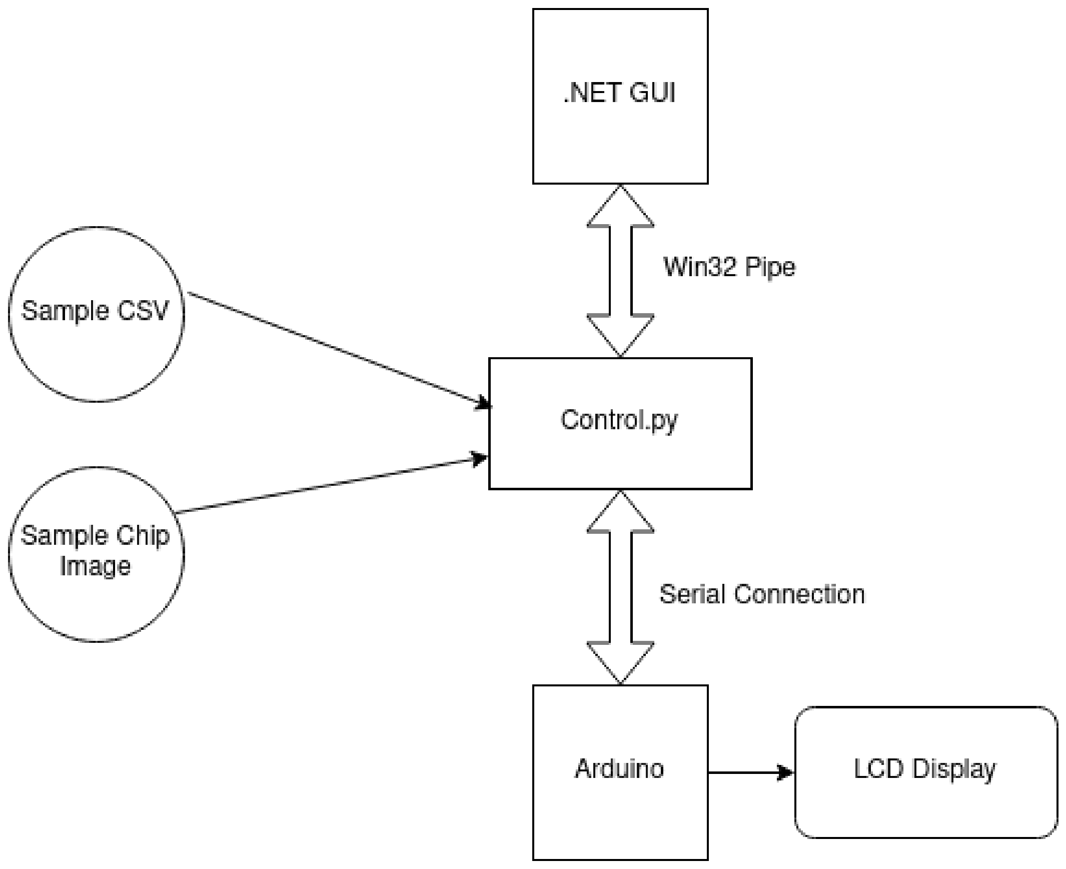
Hardware:

- Arduino Mega
- LCD display for Arduino
- Breadboard and wiring for Arduino
- Test Computer

Software:

1. .NET GUI
2. Python Programs
    a. Picture location
    b. CSV file input - [generated input]
    c. Picture Text input
    d. Serial Communication with Arduino
    e. Pipe Control with .NET
    f. Overall Control Module
3. Arduino .ino code

Set Up:



The overall setup consists of a python script which does the machine learning and main computations, a .NET graphical user interface to handle user interactions, and an Arduino which controls the hardware peripherals (eventually the movement of the machine itself). The .NET process currently starts the python control process on startup of its GUI and then communicates via a pipe to indicate which CSV file should be used as the test case. Once the start command has been given, the input CSV file is parsed and machine learning is performed on the generated chip image to find the locations. If everything passes then the machine sends commands to the arduino to perform the movement of the motors. For this prototype, those commands will be displayed on the LCD screen, but in the future this will result in movement of the motors.

Measurements:

Criteria	Completed (Y/N)
A windows application allows users to input commands and runs the python script on startup.	Y
Back-and-forth communication between the windows application and python.	Y
Using one of the generated test images, a machine learning model returns the locations of all of the chips in the traveler.	Y
Based on a preconfigured CSV file, we say what clamshells will contain what chip numbers and where they are currently located (Displayed to terminal).	Y
Prove that the arduino can perform serial communication with the computer by displaying the amount the chips needed to move to reach the different clamshell slots.	Y
Using one of the generated test images, a ML model returns the chip ID's for each chip.	N
Generate test images randomly to robustly test image recognition.	Y
Solidworks assembly will show construction of the chassis of the machine as well as the machine’s x, y, & z motions.	Y

Results:

1. **A Windows application allows users to input commands and runs the python script on startup.**

The windows application is essential to this project in order to give users a degree of control over the packing process. It was built using the .NET framework in C#. We wanted it to closely resemble the software applications already in use at NanoView Biosciences for cohesion. To this end, a vertical tab system was added on the left side of the application. Some of the components included in the GUI include a button to upload CSV files. Once uploaded, these are then displayed in a table view which also allows for editing directly in the GUI. Eventually, this will allow the user to directly change the CSV data before sending it to the python script. Another button pipes the CSV path to the backend, allowing it to begin the packing process. The second tab currently contains a graphical representation of the traveler. This will be updated visually to more closely resemble the traveler, and also to take in the proper input data from the official CSV formats once we get those, and display different types of chips using different colors.

2. **Back-and-forth communication between the windows application and python.**

Essential for the purposes of allowing the user to start and stop the machine, as well as allow communication back to the GUI to indicate various states the machine may find itself in. Some examples may include indication of chip positions not matching the input CSV file, some kind of mechanical error in the machine, or a simple message to indicate that packing has finished successfully. Currently, these various messages have not been implemented, but the ability to send strings back-and-forth between the windows form application and python script is. This should make implementing various messages extremely easy in the future.

3. **Using one of the generated test images, a Machine Learning model returns the locations of all of the chips in the traveler.**

The python image recognition class was able to take an input image with no other information given. From that it was able to draw a bounding box on the image designating where it thought chips were placed. Three images were tested in the testing procedure, all of which had been randomly generated and not previously tested against. For each image, the chip locator program was able to recognize all of the placed chips. This is an essential component of our project as we will be using machine vision to place the arm, chip, traveler, and clamshell in 3D space. Some of the complexity of this problem currently needs to be solved. The machine learning part of this computer vision model is meant to help generalize the scanning process. However, it was only partially implemented for this model with many of the parameters being set beforehand for the particular lighting values of the generated test images.

4. **Based on a preconfigured CSV file, we say what clamshells will contain what chip numbers and where they are currently located (Displayed to terminal).**

This was accomplished. We need to be able to read a CSV file and work with its contents as our client actively uses a CSV to keep track of the chips in the manufacturing steps prior to that which we are optimizing. We were able to additionally generate the CSV file to exactly match a given generated image. This added complexity helped us because it gave a visual representation of which chip in the traveler was being transmitted to the motor control at a given time. We were also able to have the CSV file that was read be incorporated as a user configurable setting through the GUI program.

5. **Prove that the arduino can perform serial communication with the computer by displaying the amount the chips needed to move to reach the different clamshell slots.**

While the mechanical construction is still being completed, it is essential that we test the electrical connections to make sure that we can smoothly transition to programming the machine to move. In the future, an Arduino Mega will be used to control the motor controllers, which will in turn move the motors of the packing robot resulting in x, y, and z motion. This means that we need to be able to reliably send commands to the Arduino over the USB serial connection as we calculate the coordinates of the chips, traveler, and clamshells using machine learning on the PC. Therefore, for this prototype we decided to test our ability to communicate with the Arduino by sending sample commands to trigger a function on the arduino. In the python control script we simulated the difference between two locations on the coordinate plane (how much the arm needs to move) using the coordinates of a chip from the test image and a random clamshell location. This difference was sent over serial as arguments, along with a function name. On the Arduino side this function name was used to parse the arguments, and then the appropriate message representing the chip and the difference was displayed on a peripheral LCD display. This allowed us to prove that our system worked in real time during testing without any issue, demonstrating reliable serial communication. In the future, we will expand this communication to handle more functions as we delve into programming the motor control for the machine itself.

6. **Using one of the generated test images, a ML model returns the chip ID's for each chip.**

This part of the test plan wasn't accomplished. We ran into issues and time constraints when developing this testing procedure (described in the section below). This is an important item to accomplish soon for our project, as it impacts our ability to accurately identify chips.

7. **Generate test images randomly to robustly test image recognition.**

This was demonstrated through the various possible configurations that the test could be run with. While this is not a part of our final product, it was an important step allowing us to more robustly test the software. The generating script takes a png of the traveler. Then it takes a png asset randomly from a list of our 4 different chips. It places a random number of chips in random locations. These gaps are meant to imitate the gaps that are left in the traveler after the testing procedure. Then the script takes this new combined png and rotates it a random number of degrees between -10 and 10. This rotated image is then scaled to be representative of the size of the traveler to our testbed and placed randomly on a grey background. The rotation and placement steps are meant to represent the human error that might exist when the traveler is placed. This grey background image is then saved as either a training image or a test image depending on a configurable split ratio. The script generates N images.

8. **Solidworks assembly will show construction of the chassis of the machine as well as the machine’s x, y, and z motions.**

We were successful in showing the chassis of our machine through SolidWorks CAD software. The 80/20 frame which we will be using was shown in the CAD model, as well as how we will achieve motion in the x and y direction through our machine connections. We were also able to successfully demonstrate how motion in the z direction will be achieved through the use of a linear actuator attached to the chassis. This is an essential component to complete so that we can begin building the packing machine.

What's Next:

Graphical User Interface

One of the biggest issues relating to the GUI is the fragility of the pipe. It can break if either the graphical side or backend script fails to close the pipe on the previous run. Therefore, if any errors occur, even if they are unrelated to the pipe, the pipe will still end up breaking. This can sometimes make debugging difficult since you may not always know where the error is coming from. For improvements, we could generate a new random pipe name on every run of the script. Additionally, the pipe completely blocks access to the application if it is waiting for data from the other end. This is clearly suboptimal as it disallows the user from performing any actions, or even from closing or even minimizing the program. Making the pipes non-blocking would be a significant improvement.

ML models:

We ran into issues with the text scanning part of our prototype. The issues that we ran into were these.

a. **Initially the problem seemed simpler than it was**

   At first we thought that any image of the chips would work for our testing plan. However, because of all the image processing that is done on the image a higher resolution would have likely worked better. When we attempted to take higher resolution images we learned that the reflective nature of the chips makes them difficult to properly light directly when the camera casts a shadow over the chip. This made us regress to our original images.

b. **We were limited in our testing data**

   Our only examples of chip numbers came from our clients' demo chips. We only have 4 chips which do not contain the complete number set.

c. **We were limited in our validation data**

   Additionally to create our Nearest Neighbor pair we had to synthesize images to approximate the chip locations. This meant that our neighbor references differed from the actual images because we didn't have the font that was used to create the numbers on the chips.

d. **We wasted time on the wrong approach**

   Our approach might have also worked better if we had bit the bullet and implemented a CNN for this task. This would have required us to generate a large number of test images however. We could have also used the actual numbers scanned to create our mean vectors in Nearest Neighbors. This would have improved accuracy but we were worried about over-fitting our model which might actually not be as big a problem for this application as it normally is.

All of these factors lead to our machine learning classifier to not be working on the day of testing. However, moving forward, these failures will help us quickly develop a new text classification schema. Additionally we will change many of our configuration values for image processing to either be determined by machine learning using a cost function or in an automated fashion with something akin to white balancing.

Object Location Scanning:

Our image scanning model can currently only identify chips. Going forward we plan to generalize this so that the traveler, any clamshells and the arm are included in this identification process.

Control Loop:

As we start to get our hardware, we plan to start implementing our motor control methodology. There are undoubtedly many unsolved challenges with this still to come.

Error Checking:

One of our main functionalities that we have yet to test is the error checking part of our packing machine. We will be scanning the chip numbers and computing their relative locations. Then we can check this against the chip numbers given in the CSV and their number in the CSV to make sure that everything matches up as desired and that there was no human error in the previous steps which could lead to error in packaging.

Mechanical:

Going forward, we will have to start building a physical model of our mechanical system. We were not able to show progress towards a physical model due to the many issues which have come up throughout the semester. The design of our system and the speed at which we will be able to develop a physical prototype will be based on a few different things.

a. **What NanoView May Provide**

   After not having received an email from our client for around a month, we received a message last week from another NanoView employee who is a senior engineer at the company. He said that NanoView would be able to provide a machine which is able to pick and place objects. We would have to determine if our device could be used for our purposes, but if it can, we wouldn’t have to purchase many additional parts. We will be visiting NanoView again this upcoming week in order to see the device and ask questions about its capabilities.

b. **Part Purchasing Costs and Shipping Times**

   If our team can use the device kit provided by NanoView, it is likely that we would still have to buy a few additional parts. In terms of cost, we would have to make sure that we limit our expenses. We have made two big purchases so far, with issues experienced for both, which will be discussed in the next section. With NanoView having already purchased a kit which we can potentially use, this expense limit would almost certainly not be an issue. If not, we would have to make more adjustments. In terms of shipping time, even if we needed only a few products, it would likely take time to get to us due to supply chain issues. Any new orders we make will have to be done as soon as possible.

c. **Effective Communication with NanoView and our Instructors**

   As mentioned above, we have run into different issues due to not having more communication with our client. We have not heard from the President of NanoView, David Freedman, in over a month, and we were unaware that they would be able to provide a kit for us until this past week. We hope to ensure strong communication with NanoView for the rest of the design process. Another hope we have is to effectively communicate our shipping issues with our instructors. Our largest purchase to date was our purchase of motors and stepper drivers, which ended up being stolen. We also have been unable to receive additional motors through our Amazon purchases with the ECE department. By explaining to our instructors the circumstances we are in, we hope to potentially receive reimbursements and clarification on next steps we should take.

Appendix A: Additional Information from Test Plan

Pre-testing Set Up Procedure:

- Connect the arduino setup (wired and flashed before testing) via USB to the computer.
- Open the lab computer, and open the file location of the GUI .exe file "NanoPack UI (draft).exe" and the solidworks model.
- Open powershell to the path of the python control module (/dev/first_prototype/)
- If needed run the script: \dev\generateTestImagesMacro\create_random_Traveler_and_chip_and_clamshell_placement.py to generate the test images if needed

Testing Procedure:

- First Test: Test software stack with Python run manually
  - Open "NanoPack UI (draft).exe"
  - Use the first button to locate the CSV file
  - From powershell run the command "python Control.py"
  - Press the second button on the GUI to begin packing
  - Verify that output on Arduino, Python, and GUI are correct
- Second Test: Test software stack with Python run from GUI
  - Follow same steps as first test, except omit the powershell command
  - Verify output on Arduino and GUI (python cannot be viewed).
- Third Test: Demonstrate the solidworks model
  - Open up the model and demonstrate the x, y, z motion of the chassis of the mechanical design.