Pick and Place Machine

Wenjun Steven Sun
Kanyon Edvall

Fall 2018

ECE 395: Advanced Digital Projects Lab
University of Illinois at Urbana-Champaign
# Table of Contents

Abstract 2  
Terms and Keywords 3  
Overview 4  
   Motivation and Project Scope 4  
   Design 4  
   Implementation 5  
   Going Forward 7  
Brushless DC Motor Controller 8  
   Motivation and Project Scope 8  
   Design 8  
   Implementation 9  
   Going Further 16  
OpenPNP Server 16  
   Motivation and Project Scope 16  
   Design 16  
   Implementation 16  
   Going Further 17  
   OpenPNP to ODrive Interface Code 18  
Project Pictures 20
Abstract

The goal of our ECE 395 project was to create a pick and place machine exceptionally better than the majority of open source projects available and significantly cheaper than similar solutions available on the market. To achieve this, we created a brushless DC motor controller and wrote an interface to connect the controller to the OpenPNP software. We were successful in realizing our design but a number of improvements will be made to the system next semester.

The completed PNP prototype!
Terms and Keywords

- Pick and Place - A machine designed to pick up surface mount components from part reels and place them on a bare PCB for soldering.
- P&P / PNP - A pick and place machine
- SMT / SMD - Surface mount technology or surface mount devices
- BLDC - A brushless DC motor
- PCB - Printed circuit board
Overview

Motivation and Project Scope

The idea for building a pick and place machine came from the company we work for, LifeFoundry, wanting an in house PNP. The problem with this is the majority of pick and place machines cost in the ten to one hundred thousand dollar range, much beyond our budget. Lower end PNP machines do exist but have significant drawbacks, they typically user stepper motors and have no positional feedback. These two caveats make low cost PNP machines difficult to use and very slow to operate.

To remedy these drawbacks, we decided to create a PNP machine using brushless DC motors (BLDCs) with positional feedback. BLDCs are known for being complicated to control and in order for us to achieve our design we needed to create a motor controller. We would also need software to run pick and place routines and to perform basic verification of component placement. Finally, to actually implement our design, we would need to build the physical parts, both mechanical and electrical, of the pick and place system.

Design

This section will cover the design of our pick and place project as a whole. For a more detailed design on individual components see their respective sections. The scope of our proposed project was rather large and designing everything we needed from the ground up would have taken longer than the semester we had to work with. Thankfully, we were able to pull resources from a number of open source projects to speed up our development. To start the overall pick and place design we researched existing open source projects. Neither of us have a mechanical background and we figured the mechanical design of existing PNP systems would work well enough with a few modifications. We were mainly interested in overhauling the electrical system and the drive system. Eventually the project we decided to base ours off was called DIY Pick and Place (available here: https://hackaday.io/project/9319-diy-pick-and-place). The mechanical design seemed robust and the extruded aluminum profiles made modification and attaching additional equipment easy.

Next we focused on the software we would need to run our pick and place machine. We discovered an open source package called OpenPNP (available here: http://openpnp.org/). This software gave us a graphical interface for controlling the machine, allowed us to import PCB files for component placement, and actually carried out automated control of the machine. The only downside of OpenPNP was that it did not support the firmware of the motor controller we used, requiring us to write an interface between the two.

Finally the most important aspect of our project, the motor controller. BLDCs are quite difficult to control when compared to most other types of motors such as DC and stepper motors. Designing a BLDC controller completely from scratch is quite an
undertaking from a software, hardware, and firmware point of view. Luckily we found an excellent open source project called ODrive (available here: https://odriverobotics.com/#odrive). This project provided a good base for us to start development, both in terms of hardware and firmware. Our planned design for a new board was to shrink it to approximately one third of the size of the original board, integrate support for DJI’s motors with built in encoders, and add support for CANbus to the controller.

Implementation

This section will cover the implementation of the overall pick and place machine. For component specific implementations see their respective sections. To fully realize our goal of creating a faster, more robust pick and place machine, we needed to integrate each of the components we developed. After laying out our initial design requirements and goals, we started creating the frame for our PNP. We based our mechanical structure off the DIY Pick and Place projected noted earlier (reference here: https://hackaday.io/project/9319-diy-pick-and-place). We used 20x40 and 20x20 extruded aluminum profiles we purchased from the machine shop and had cut to length. For linear rails, a set was purchased from eBay and cut to fit our application. The type we used were called Miniature Linear Sliding Rails and cost about $70 total to purchase. A sheet of MDF wood was selected for the base of the pick and place but we ended up not attaching it.

Next up we purchased motors. We had one BLDC from an electric longboard we had built a couple years before, but needed a second. HobbyKing has a good selection of a bit overpriced motors but the shipping is pretty fast so we went there. When selecting a motor just about any BLDC will work but we like the Turnigy Aerodrive SK3 series a lot. The main difference between various SK3 models is the Kv value. This value is a measure of the rate at which a motor with no load would spin per one volt applied to it; it’s a bit of a strange measurement with units of RPM/Volt. In general, the same series of motor will have a higher speed with lower torque for larger Kv values, and a lower max speed with higher torque for a low Kv value. For our application the motor’s Kv isn’t all that important but selecting one with a Kv in the range of 200 to 400 works well. The second motor we got was a Turnigy Aerodrive SK3 - 5065-275KV, an SK3 model with a Kv of 275. The motor actually ended up being a bit overkill for our PNP but worked well nonetheless.

We now needed a way to mount the new motors to our pick and place. The original design used stepper motors so the motor bracket hole patterns were laid out for Nema 17 steppers. Because we had limited time and cared more about the control system proof of concept, we decided to laser cut our mounting brackets from plywood. This allowed us to easily modify the original motor mount design to accommodate our motors and was very quick to manufacture. This design choice did have a significant drawback however that will need to be fixed in a future version. The plywood is not rigid enough to prevent motor wobble, especially under high stress like when moving. In addition, our belt was unable to be properly tightened due to the poor strength of the wood used. That said, they did work well enough to get a working design prototype. All
that was left now was belts, pulleys, and mounting everything together. We used a GT2 timing belt and standard GT2 pulleys that fit our motor’s shaft. Rails were mounted using T-Nuts and M3 screws. Additionally, 90 degree, 20 series connectors were used to join the aluminum profiles together. We 3D printed a mount for the Y-carriage that also allowed us to lock the belt in place. This was a design choice made for simplicity but makes belt tensioning a bit difficult and could be improved.

We were now able to add the motors and wire up the electrical system. Initially we used the custom designed ODrive board we created. However, after making revisions to the board, the shipment from China was delayed resulting in our new boards not arriving on time. As our new boards can only control one motor, and we only had one board for testing, we had to switch back to the original ODrive board for our demo in order to run both axes at the same time. The required us to mount external encoders on our motors to track shaft position, and it forced us to mount the ODrive on the side of the machine instead of as close to a motor as possible. We still used our custom ODrive board up until we began testing with two axes simultaneously and it performed very well. One motor was attached to the main gantry (the X axis) and moves the head along the length of the PNP. The second motor was attached to the secondary gantry (the Y axis) and moves the head across the bed of the PNP. The original ODrive board is capable of driving two motors with a single board so we connected a motor to channel 0 and channel 1. The ODrive has quite a high current capacity at 80A so we recommend using wires with AWG 18 or lower. We had to bring in a separate DC power supply as the lab supplies were limited to 1A. We used a supply with a 24V output limited at 10A and connected this directly to the ODrive. The ODrive automatically regulates down to 5V for its on board logic in addition the the 24V, high current motor supply lines. Now the hardware and electrical systems were ready to go!

After finishing the mechanical assembly and attaching the motors and belt it was time for an initial test. We started with a single axis to test whether control was functioning at all. To facilitate this low level control, we took advantage of the ODrive’s on board USB port and the ODrive Tool. The ODrive Tool is a small Python utility that provides a basic interactive shell to control and test the ODrive board (available here: https://docs.odriverobotics.com/odrivetool.html). For basic movement testing we used the following command: ‘axis0.controller.pos_setpoint = <encoder_counts>’ where <encoder_counts> is the desired position in terms of the encoder’s output. We got movement! At this point we could successfully command an axis to move to a specified location. The final steps were now to connect this to OpenPNP to run automated routines instead of hand controlled movements. This required us to write a custom interface server as described below.

With the server implemented and running on the controlling computer, our custom pick and place was ready to go with one major component missing; the pick and place head. Pick and place machines use a pneumatic nozzle to pick up parts. They pick a component from a reel of parts, move to the desired location, and release air pressure to place the part. This makes building them an exercise in mechanical engineering and air control, something neither of us have knowledge about or the time to learn. Fortunately for our project, pick and place heads are relatively cheap, easy to buy, and work great, saving us the effort of designing and machining one ourselves. We
wanted to make sure we had a fully moving prototype before spending the $500 to purchase this part so we were a bit delayed in ordering it and it did not arrive in time. For our demo we put a placeholder where the head would be and simulated moving it to various pick up and placement locations but obviously weren't able to actually pick up or place parts. If you’re interested, the referenced pick and place machine has a fully functional head that can be seen in the demo video.

Going Forward

Overall we are happy with the progress on our project this semester. We were able to create a custom ODrive board with an impressively small form factor still capable of handling 40A, build a functioning prototype of the mechanical and electrical systems with full movement supported, and integrate our system with OpenPNP for easy, robust control. We even achieved a repeatable movement accuracy of 0.05mm as measured by the ECE Machine Shop. Steven will continue working on this project next semester with the goal of having a complete, usable system we can actually use to manufacture our PCBs! There are several areas we see can be targeted for improvement.

First up is the mechanical structure. The 20x20 profile is certainly rigid enough but was not fastened well enough to stop vibrations. In addition rubber feet are a must add. The main problem, however, is the motor mounts. The plywood we found in the Open Lab was convenient and free but is also quite flimsy. We would like to 3D print and mill proper motor mounts that give us much greater structural integrity. This will help with axis wobble, achieving and maintaining proper belt tension, and prevent the X axis from slipping during fast movements. We also need to improve the encoder mounts and the shaft extensions driving them. They are currently command stripped in place and have paper wedged in them to tighten them against the shaft. As you might imagine, this is not a very robust system and began to fail after we ran the machine for awhile. Integrating our motors with built in encoders we be a potential solution to this.

Next we would like to migrate from the original ODrive board to our custom ODrive boards. This will allow us to place the board directly next to a motor essentially allowing us to build small servo motor. Changing to our board will also allow us to use our DJI motors with built in encoders, potentially allowing for higher accuracy movements. It will certainly be more robust than our command strip encoders.

Lastly, we obviously need to integrate the PNP head to actually use the PNP machine. As this is an off the shelf component, integration should be relatively simple but will probably require some firmware additions if we want the ODrive to fully control this machine. There will need to be pneumatic tubing adding and some sort of compressor to drive it. Our plan for adding the head will become more concrete as we finish researching how the PNP head is typically integrated.

These are the main areas requiring improvement and we would like to complete all of them over the next semester. If possible we may add additional features and improvements if there is time.
Brushless DC Motor Controller

Motivation and Project Scope

We originally found the ODrive project while looking for a robust, low-cost way to control a brushless DC motor (BLDC). While the original ODrive was an excellent starting point there was one major problem with it; it was huge. For the majority of our applications, we want a small motor controller that can be paired with a single motor and ideally be mounted near it, essentially creating a small, low cost servo motor. The original ODrive’s sheer size prevents this. In addition, the original ODrive is rather overkill for our project in terms of available power. A single ODrive board is capable of simultaneously driving two motors independently and can use up to 80A. Nothing we are controlling requires that amount of current, and we prefer using a single motor with a single board.

A second problem we faced was integrating some DJI motors we like with the ODrive. The original ODrive board did not work with the encoder integrated into the motor requiring us to have an external board act as an interface. This was not a very robust design and certainly wouldn’t work long term.

These application specific shortcomings of the original ODrive project led to us developing our custom ODrive board, the Mini ODrive. The goal of this project was to create a small form factor board that integrated the additional hardware needed for controlling the DJI motor. The new board design would have a lower current rating and only drive a single axis, but this was suitable for our design requirements and was a necessary tradeoff.

Design

ODrive is an open source design that turns hobby BLDC motor into a precise servo motor. The benefit of this is we can get a super low cost, fully customizable, fast, precise, closed loop actuator. This is a really general project that can benefit a lot of people who want to have a motor related project.

Odrive is a two channel, 80A driver that you can buy for around $120 online. However Odrive is really big and only has two channels. So if someone wants to build a machine that has odd number of motors, they will be wasting money and space. We redesigned a much smaller version of ODrive that has half of the current and half of the available channels, allowing it to save space on the project and reduce running wires since each Mini ODrive can be placed to almost directly on the motor it’s controlling.
Implementation

After laying out project goals for our custom revision of the ODrive we started designing it. Altium Designer was used for making the PCB. Pictures 1 and 2 (below) show the schematic of our most recent PCB design.

Picture 1
We were able to successfully shrink the Mini ODrive to a fraction of the original’s size! In addition to reducing the size, a connector and accompanying hardware was added to integrate directly with the DJI motor and the onboard encoder. All heat producing components were placed on the bottom of the board to allow it to sink heat into a case or large heat sink.

One problem we ran into when actually using our Mini ODrive board was the placement of encoders on some of our motors. One gimbal style motor we tested with forced us to mount the encoder to the side of the shaft. Having the encoder off center cause a very nonlinear encoder readout that ruined positional control. To solve this problem we added a sort of manual calibration for characterising the motor and encoder sensor placement. To do this we created a lookup table in the ODrive’s firmware and populated it by completing a full rotation of the motor and reading the encoder value at each position. Then when the motor moved, the lookup table was used instead of the actual encoder reading. This allowed us to create an excellent linearization of the encoder’s reading and fixed our positional control instability. The code to do this was pretty quick to write after we figured out how to use this method. Part of the code is shown below (Picture 3).
uint16_t decode(uint16_t in)
{
    int i=0;
    while(in>cal_table[i])
    {
        i=i+10;
        if(i>1036)
        {
            i=1036;
            break;
        }
    }
    while(in<cal_table[i])
    { i--; 
        if(cal_table[i]==16328)
            return 16328;
        uint16_t a1=cal_table[i];
        uint16_t a2=cal_table[i+1];
        double percent=(float)(in-a1)/(a2-a1);
        return (i+percent)*14*1.125534;
    }
}

Picture 3
The following pictures show the encoder’s output before and after linearization.

Picture 4

Picture 5
Picture 4 (above) depicts the original, unaltered encoder output. Picture 5 (also above) shows the new, corrected encoder output after the lookup table is used. A linear fit (the blue line) is show for reference.

While implementing the Mini ODrive we went through a couple revisions. The initial design had several hardware design bugs including a missing trace that had to be jumped. The first revision schematic is shown below (Picture 6).

After receiving the initial testing board from China we noticed problems during testing. A second revision was quickly made but was unfortunately delayed at US Customs and did not make it in time for our demo. The new and most current revision (Picture 7) not only fixes various problems but also shrinks the board even more and adds better connectors for integrating the board directly with our hardware. The top and bottom of the second revision PCB (newest) is shown below.
Finally, Picture 8 (below) depicts both the first and second revision of the Mini ODrive after being soldered and connected for testing. The second revision is on the left, the initial design is on the right.
Going Further

After going through a couple revisions of the mini ODrive board we think we have a fully function potentially final revision. Barring discovering any hardware bugs, future updates to the ODrive will most likely be firmware related. While we currently don’t have a well developed plan, these updates may include adding support for actuating the pick and place head directly from an ODrive board.

OpenPNP Server

Motivation and Project Scope

The OpenPNP server was necessary to connect the ODrive to OpenPNP. The project encompassed receiving GCode from OpenPNP and translating into commands sent to the ODrive over USB.

Design

We decided to write a Python program to translate between GCode and commands the ODrive could understand and accept over USB. The program was designed to connect to the ODrive using the ODrive API and run a local server to listen for incoming commands.

Implementation

We used the socket, signal, sys, and odrive libraries to create our OpenPNP to ODrive interface. A local server was created using a socket that listens for incoming commands from OpenPNP. When a command is received it is translated to an ODrive command and sent over using the odrive library to be executed by the board. Pressing Ctrl+C uses a signal handler to kill the server.

Currently, recognized GCode commands are stored in a dictionary. Any unrecognized commands are simply rejected. As of now supported commands are “G21”, which sets the machine to millimeter mode, and “G0” which is a movement command. When a movement command is issued, the GCode is parsed to find the velocity, X axis position, and Y axis position. The server then translates these to ODrive commands in the form of “odrive.axis1.controller.move_to_pos(float(x)*100)”. The necessary commands are then sent and the server waits for the movement to complete.
Going Further

The last step in completing our PNP to ODrive interface server is potentially adding translation for the GCode commands that pertain to the PNP head operation. As we haven’t received the head yet we aren’t sure whether this is necessary and if so what it will entail.
OpenPNP to ODrive Interface Code

The OpenPNP to ODrive interface server code is included below for reference.

```python
import socket
import signal
import sys
import odrive

def signal_handler(sig, frame):
    print('You pressed Ctrl+C!')
    conn.close()
    GDriver.close()
    sys.exit(0)

signal.signal(signal.SIGINT, signal_handler)

odrive = odrive.find_any()
GDriver = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
GDriver.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
GDriver.bind(('127.0.0.1', 8891))
GDriver.listen()

def CMD_G0(data):
    x_valid = 0
    y_valid = 0
    velocity = -1

    loc_vel = data.find('F')
    if (not loc_vel == -1):
        vel_data = data[loc_vel:]
        vel_data = vel_data[1:vel_data.find(' ')]
        velocity = float(vel_data)

        if (velocity < 50):
            print('velocity too slow, limiting to 5%')
            velocity = 50

    loc_x = data.find('X')
    if (not loc_x == -1):
        x_data = data[loc_x:]
        x = x_data[1:x_data.find(' ')]
        x = float(x)
        odrive.axis1.controller.move_to_pos(float(x) * 100)
        x_valid = 1
    else:
        x = 0

    loc_y = data.find('Y')
    if (not loc_y == -1):
        y_data = data[loc_y:]
        y = y_data[1:y_data.find(' ')]
        y = float(y)
        odrive.axis0.controller.move_to_pos(float(y) * 100)
        y_valid = 1
    else:
        y = 0
```
# if x or y move valid then wait for xy move finish

```python
while (y_valid or x_valid):
    x_loc = odrive.axis1.encoder.shadow_count
    y_loc = odrive.axis0.encoder.shadow_count
    if x_valid:
        delta_x=x_loc - float(x)*100
    else:
        delta_x=0
    if y_valid:
        delta_y=y_loc - float(y)*100
    else:
        delta_y=0
    #delta_x and delta_y can be positive or negative
    # print("waiting for XY to finish, dx:{},dy:{})\n.format(delta_x,delta_y)
    if(abs(delta_x)<50 and abs(delta_y<50)):
        break
    print("MOVE to X:{} Y:{} with vel: {}\n.format(x,y,velocity))
```

def CMD_G21(data):
    print("set mode to milimeter")

GDict = {
    "G0": CMD_G0,
    "G21": CMD_G21
}

print('Gcode Interpreter running')
print('Waiting server to connect')

conn,addr=GDriver.accept()

print(conn)
print('Server connected. Addr:',addr)

while(1):
    server_msg=conn.recv(1024)
    if(server_msg==b'\n'):
        continue
    print('msg: %s' %server_msg)
    # if(server_msg)
    data = server_msg.decode('ascii')
    GCODE=data[:data.find(' ')]
    GCODE_FULL=data[:data.find(';')]
    if GCODE in GDict:
        GDict[GCODE](GCODE_FULL)
    else:
        print("not supported {}\n.format(GCODE))
    conn.send("ok
n\n.encode('utf-8'))

conn.close()
GDriver.close()
Project Pictures