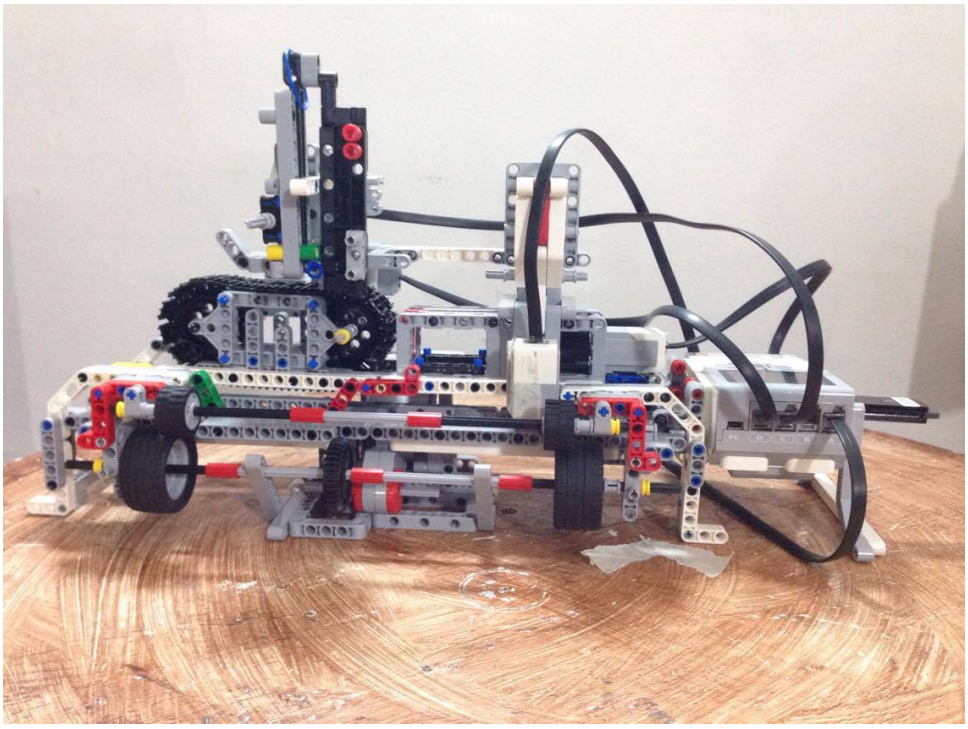
Joy Gu, Alan Zhang
PKP Lego Group 4
Dr. Gábor Csányi, Mr. Matthew Coates
Summer 2016

PKP Lego Robotics Project Report

Dotty the Copier



Introduction

For our project in the PKP Lego Robotics Course we wanted to design and build a scanner/copier machine. Our concept was mainly a large mechanical challenge that required several different prototypes of parts and adjustments until it satisfied the behavior we specified. However, the code-writing in Matlab also became a challenge to fully integrate all the our mechanical functions together with accuracy. After the last four weeks of work, we are proud to present our machine, Dotty the Copier.

Original Plan

Our original project plan was to create a machine that scans a design pattern consisting of white and black grids/pixels, and copies it on the same piece of paper. We planned to have two attachments that would work together: a sensor that would scan the pattern, and a pen attachment that would move together with the sensor and therefore copy the pattern simultaneously. Our design incorporated a color sensor to determine if each pixel was black or white, and a pen attachment that uses a stamping motion to create dots on the paper. We also planned to scan the pattern in a z-formation (left to right, down, then right to left), and envisioned the scanner portion or the entire structure to be moving along a track to accomplish the vertical movement.

The milestones of our original plan, in order, were:

- Build a scanner to scan one row across and distinguish pixels from black/white.
- Build the pen attachment and write code for the pen/marker to move in a stamping motion.
- Integrate both the scanner and the pen movement together, and use the z-formation scanning pattern.
- (Extended milestone: if we had time) Write a script to print a design that we input.

From the milestones we originally set, we accomplished all except for the last extended one. Our machine is able to scan multiple rows, and collects readings from black and white pixels with reasonable accuracy. We have an attachment that moves a marker up and down in a stamping motion when the sensor detects a black pixel as it moves across. We integrated both tasks together and can scan an image in a z-formation pattern, although our approach is to use a paper roller to move the paper line by line, instead of the original design by using paper rollers instead of moving the entire structure.

Final Construct Documentation

Mechanical Structure:

Figure A: front view

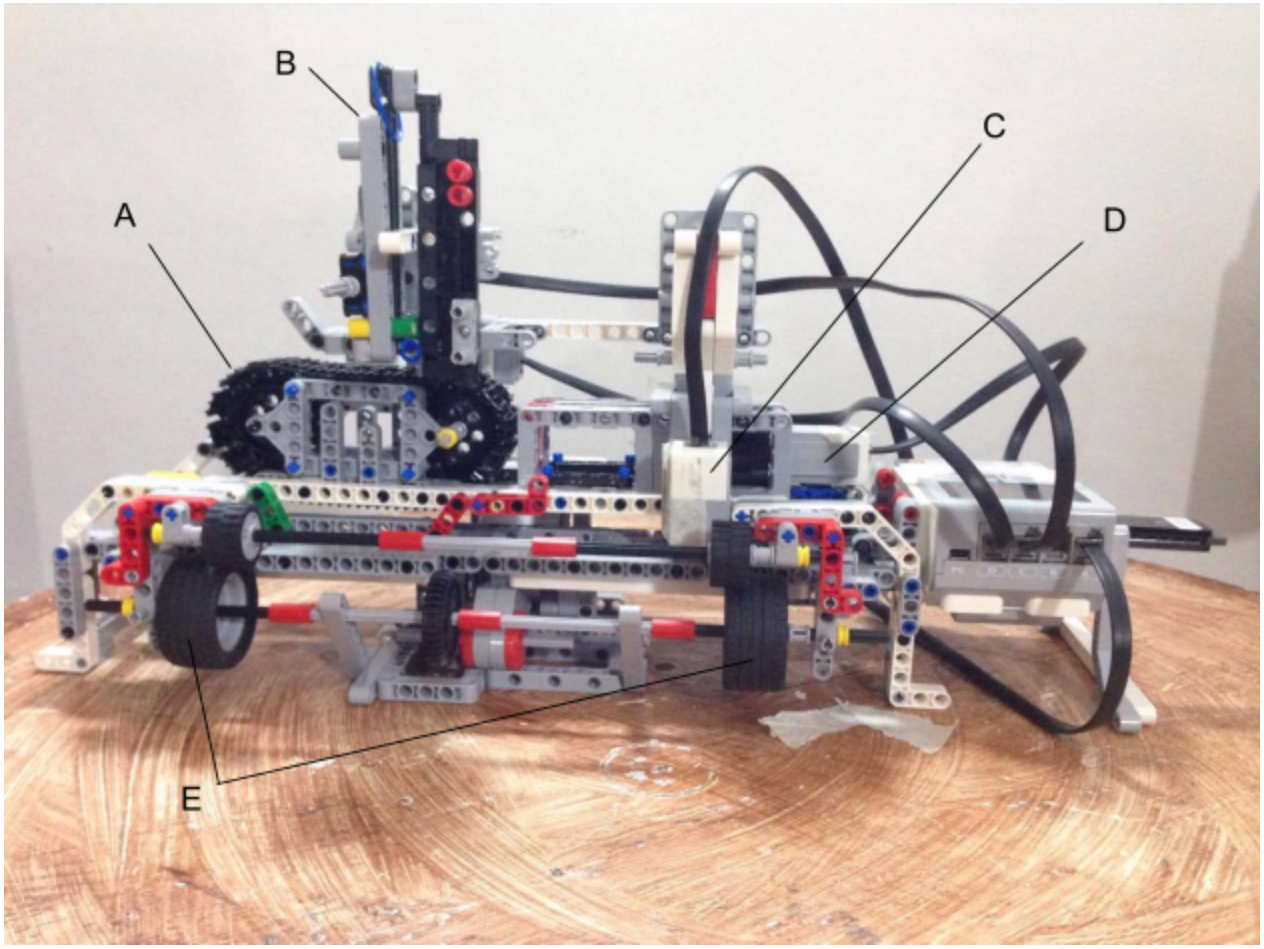


Figure B: pen attachment close-up

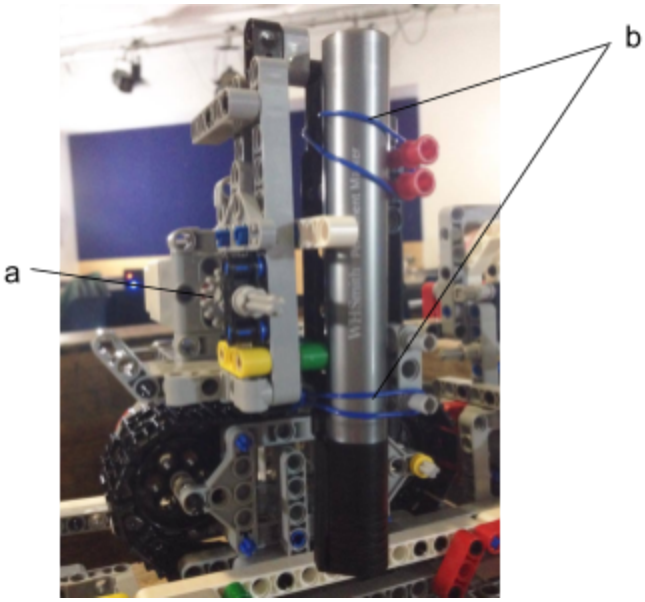
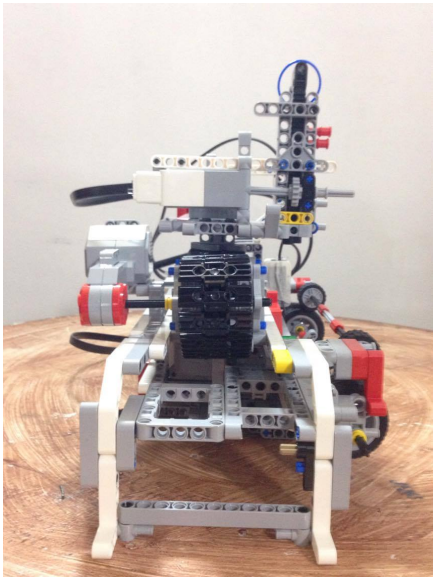


Figure C: *side view*



A. Conveyor Belt/Track

The belt is constructed by two gears holding up each of the ends of the track. We have a box-like structure that supports the middle of the belt so the top is flat. The left-most gear is attached to a motor (seen in Figure C) to move the belt and the attachment at the top.

B. Pen Attachment (reference Figure B as well)

The pen attachment is connected to the top of the conveyor belt and also has a supporting arm to the color sensor.

a. Rubber bands hold the marker to the black lego piece that keeps the marker stable.

b. The motor attaches to a small gear that rests next to a vertical serrated edge brick, which provides the linear motion as the gear turns.

C. Color Sensor Attachment

The color sensor is approximately 1 cm above where the paper rests, and the sensor arm is connected on a track so that it is stable and slides left and right without shaking. The arm is also connected to the pen attachment, and will move in line with the belt and the pen attachment

D. Touch Sensor

The touch sensor is attached on the rightmost side of the sensor track to determine whether the sensor has arrived back to the starting point.

E. Paper roller

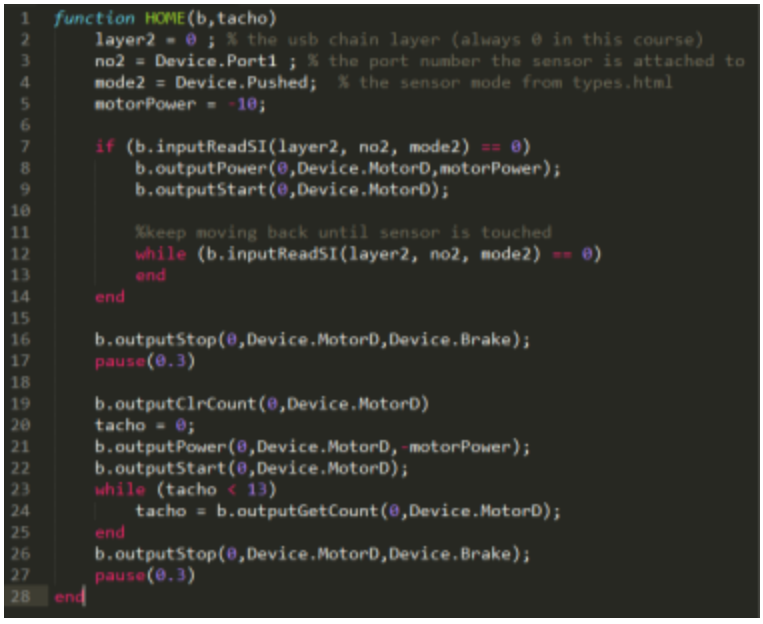
The roller consists of 4 wheels total. A larger gear is attached to the axle, and rests on a smaller gear attached to the motor. When the motor runs, the gears move the

two larger wheels on the bottom. The smaller wheels on the top rest on the top of the large wheels and rotate to provide friction and pressure for the paper to slide through.

Matlab Code:

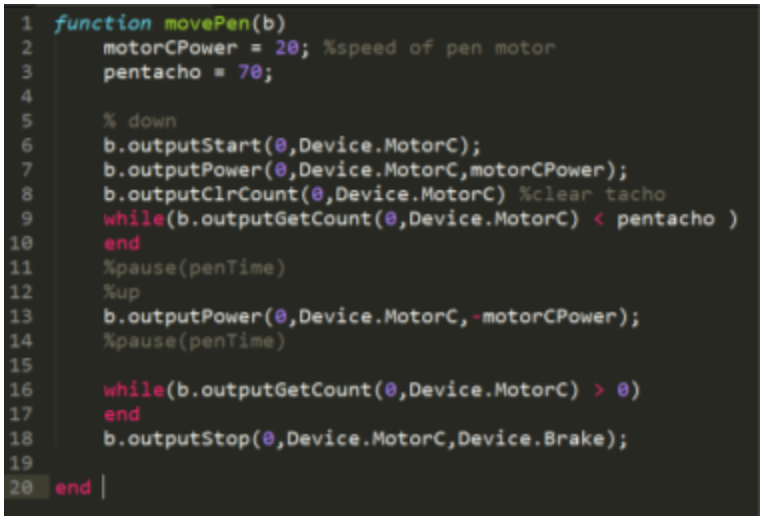
Our Matlab script consists of our main body of code and a couple functions that we call. The functions that we have written are HOME and movePen.

HOME is a function that takes in the brick object and the tachometer reading variable, and returns the color sensor to the starting position, the rightmost side of our scanner. Once the scanner hits the touch sensor, we have it move certain degrees off of the sensor before the normal movements so that the spacings can begin without being affected by the sensor arm pushing on the touch sensor.



```
function HOME(b,tacho)
    layer2 = 0 ; % the usb chain layer (always 0 in this course)
    no2 = Device.Port1 ; % the port number the sensor is attached to
    mode2 = Device.Pushed;  % the sensor mode from types.html
    motorPower = -10;

    if (b.inputReadSI(layer2, no2, mode2) == 0)
        b.outputPower(0,Device.MotorD,motorPower);
        b.outputStart(0,Device.MotorD);

        %keep moving back until sensor is touched
        while (b.inputReadSI(layer2, no2, mode2) == 0)
        end
    end

    b.outputStop(0,Device.MotorD,Device.Brake);
    pause(0.3)

    b.outputClrCount(0,Device.MotorD)
    tacho = 0;
    b.outputPower(0,Device.MotorD,-motorPower);
    b.outputStart(0,Device.MotorD);
    while (tacho < 13)
        tacho = b.outputGetCount(0,Device.MotorD);
    end
    b.outputStop(0,Device.MotorD,Device.Brake);
    pause(0.3)
end
```

The movePen function also takes in the brick object as a parameter, and it moves the pen attachment down and up, using the tachometer readings from the motor.



```
function movePen(b)
    motorCPower = 20; %speed of pen motor
    pentacho = 70;

    % down
    b.outputStart(0,Device.MotorC);
    b.outputPower(0,Device.MotorC,motorCPower);
    b.outputClrCount(0,Device.MotorC) %clear tacho
    while(b.outputGetCount(0,Device.MotorC) < pentacho )
    end
    %pause(penTime)
    %up
    b.outputPower(0,Device.MotorC,-motorCPower);
    %pause(penTime)

    while(b.outputGetCount(0,Device.MotorC) > 0)
    end
    b.outputStop(0,Device.MotorC,Device.Brake);

end
```

In our main body of code, we first define several constants that include motor power, constants for pause intervals, variables for the color sensor, and dimensions for a matrix of values (number of columns and number of rows in the image we're scanning).

```
%Constant Values
%---Scanning Variables---
%Color Sensor Variables
layer = 0 ; % the usb chain layer (always 0 in this course)
no = Device.Port4 ; % the port number the sensor is attached to
mode = Device.ColColor;  % the sensor mode from types.html
colorcons = 4; %the judgement reference for either black or white
%---Time Constant---
switchTime = 0.37; %Time for switching to the next row
readInterval = 0.005; %Time of delay for motor reading
scanInterval = 0.3; %Time for scanning each brick
%---Motorspeed Constant---
startPower = 30; %Boost speed in order to get over the resistence
motorDPower = 16; %speed of conveyor belt
motorAPower = 20; %speed of paper roller
%---Tacho Constant---
tachomax = 140;
switchtacho = 42;%degrees by which the paper roller will move every time
bottacho= 0; %bottom tachho
%User Input
userIn = '';
%---Matrix Constant---
colNum = 5;
rowNum = 14;
numScanned = 0; %how many bricks we have scanned
values = zeros(rowNum, colNum);
```

We used user input to start and finish the main functionality. When the program runs, the user either inputs 't' or 'f'. A 't' character will start the scanning process, and an 'f' character will terminate the program. This is the outermost while loop, so once the scanning/copying is done, the user can decide if he/she would like to scan another image or exit the program. After a user inputs 't', we have another while loop that keeps track of how many rows we have scanned.

At the start of this loop, we call the HOME function so the scanner begins at the correct position and the color sensor takes the first reading of the row. Within this while loop, we also have a loop for the left movement of the scanner and a loop for the right movement of the scanner.

In the loop for the left movement of the belt, we begin by starting the motor with a slight boost to overcome friction while starting the motor. While the motor is moving, we read the tachometer and divide the total degrees by the appropriate amount to match the spacings of the grid we are scanning. At each pixel of the grid, we stop and take a color sensor reading. We compare it to

```
%Belt goes forwards
while (n < colNum)
    %--speed of motor--
    b.outputPower(0,Device.MotorD,startPower);
    b.outputStart(0,Device.MotorD);
    pause(readInterval)
    b.outputPower(0,Device.MotorD,motorDPower);
    tacho = 0;
    while (tacho < (tachomax/(colNum-1))*n)
        tacho = b.outputGetCount(0,Device.MotorD);
        %pause(readInterval);
        %disp([1 tacho tachomax]);
    end

    disp([1 tacho tachomax]);

    b.outputStop(0, Device.MotorD, Device.Brake);
    pause(scanInterval);

    %scan
    reading = b.inputReadSI(layer, no, mode);
    if(reading < colorcons )
        movePen(b)
        R=0;
    else
        R=500;
    end

    numScanned = numScanned + 1;
    disp([num2str(numScanned) ' bricks scanned' ', value ' num2str(reading)]);
    n=n+1;

    %add to A row vector
    A = [A R];
end
```

a threshold called colorcons to determine if the pixel is black or white. If it is black, we call the movePen function; otherwise, we continue moving, updating the constants we are using to keep track of the number of bricks scanned thus far. All the readings are stored in an array and placed in the matrix later.

The scanning movement to the right is almost identical to the left movement loop, only the motor power is negative and the spacing calculation slightly differs.

At the end of both the left and right movements, we controlled the lower motor to move the paper through the machine so we can scan the next row. We also used the tachometer to control this movement.

```
%move bottom paper
b.outputPower(0,Device.MotorA,motorAPower);
b.outputStart(0,Device.MotorA);
b.outputClrCount(0,Device.MotorA)
while(b.outputGetCount(0,Device.MotorA) < switchtacho)
end
bottacho = b.outputGetCount(0,Device.MotorA);
disp([8 bottacho switchtacho]);
b.outputStop(0,Device.MotorA,Device.Brake);
pause(0.1)
```

Design and Challenges

Mechanical

A. Total Structure Design

Our original design was larger in scale because our goal was to scan a full sheet of paper. Due to the constraints in the number of bricks we had, we decided to scale back and scan a smaller design. Ideas that we had in the beginning also included moving the entire structure on some sort of track, so we can incorporate y-direction movement. This idea also proposed the challenge of brick quantity, so we decided against building that and instead used a paper roller concept, which will be discussed in a later section.

We began with a basic prototype that consisted of a conveyor belt and an arm with two attachments on either side of the belt, as shown in Diagram 1. In this prototype, we had an unstable base, a bumpy conveyor belt movement, and an uneven weight distribution across the arm (see Figure D) Using the prototype, we were able to pinpoint some of the mechanical problems we would encounter, and continued to work on improving each aspect of the design.

Diagram 1: Original prototype diagram (top view)

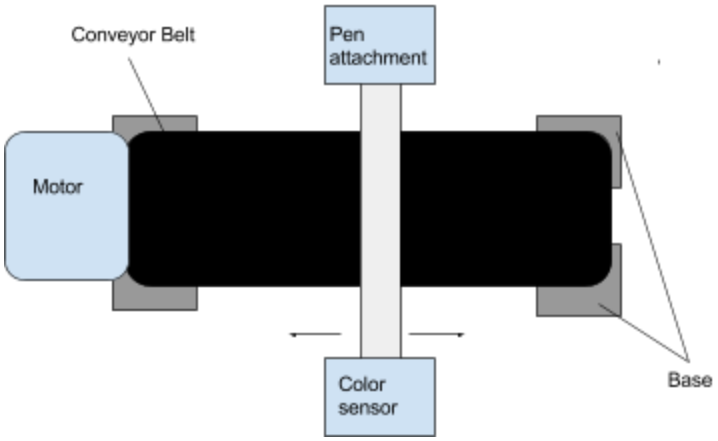
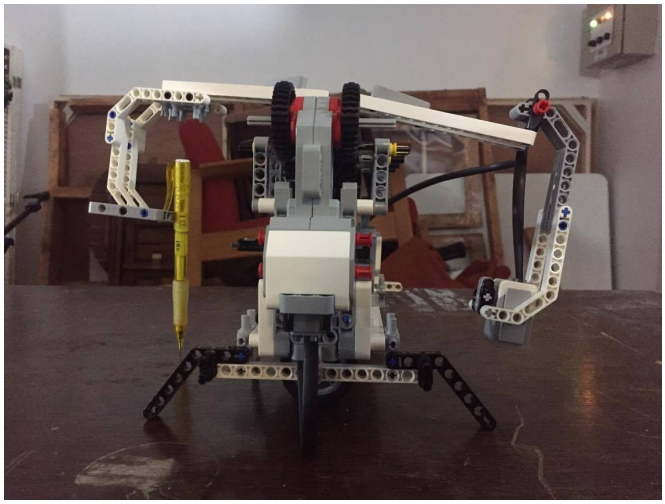


Figure D: Original Prototype side view



After working out the problems of each original part and any additional parts (see below), we came up with our final design.

B. Conveyor Belt

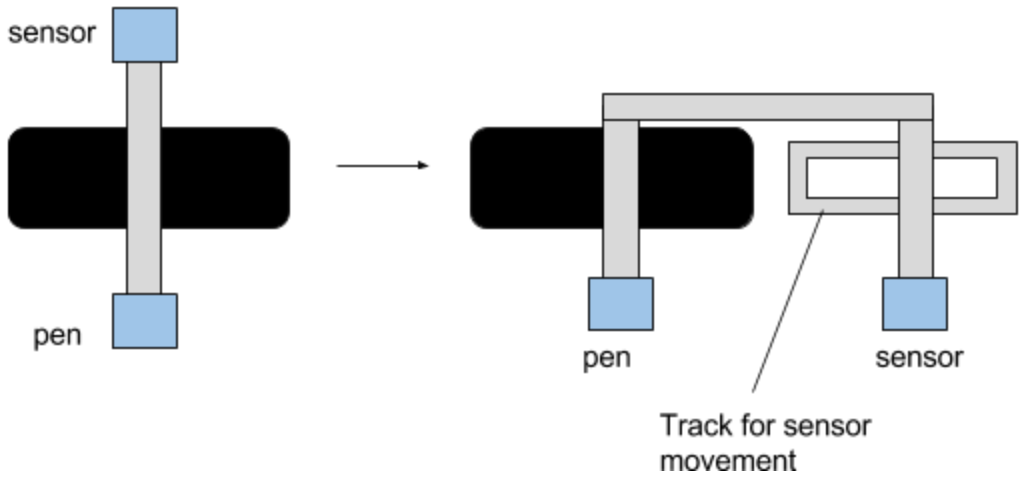
We decided to use the conveyor belt for the scanning motion because it seemed like a straightforward way of translating the rotational motion of the motors to the linear motion we needed and many real-life machines use conveyor belts to move things around. In our prototype, we held the conveyor belt up by filling the space with only gears, and using the motor and another gear to touch the outside of the belt to move it. This resulted in a very unstable and bumpy belt that often would not start moving. We attempted a different prototype of using gears and the longer serrated bricks to provide the linear motion, but found that it did not provide

enough range, and instead used this design for the pen movement. After looking at how other Lego builders built conveyor belts through Youtube videos, we took this inspiration and constructed the belt with only two gears on each side, and a box-like structure to support the middle. This allowed the belt to lay flat, providing a smoother movement than we had before. We then shortened the belt in length to conserve bricks and to downscale the project to allow for better accuracy.

C. Scanner Arm

One of our initial ideas was to make a scanner arm that would be foldable or detachable, to allow a faster transition to the printing mode of our machine, because scanning would be unnecessary for that part. However, after making an attachment that was foldable, the problem was that the distance between the scanner and the paper varied, which would lead to inaccurate sensor readings. Thus, we modified the scanner arm that was stable so the distance from the paper would be consistent. Another change we made in the design process was moving the sensor portion from hanging on one side of the conveyor belt, like in our prototype, to moving parallel to the conveyor belt on a track (see Diagram 2) to create a more stable structure.

Diagram 2: Sensor Design Change



D. Pen Attachment

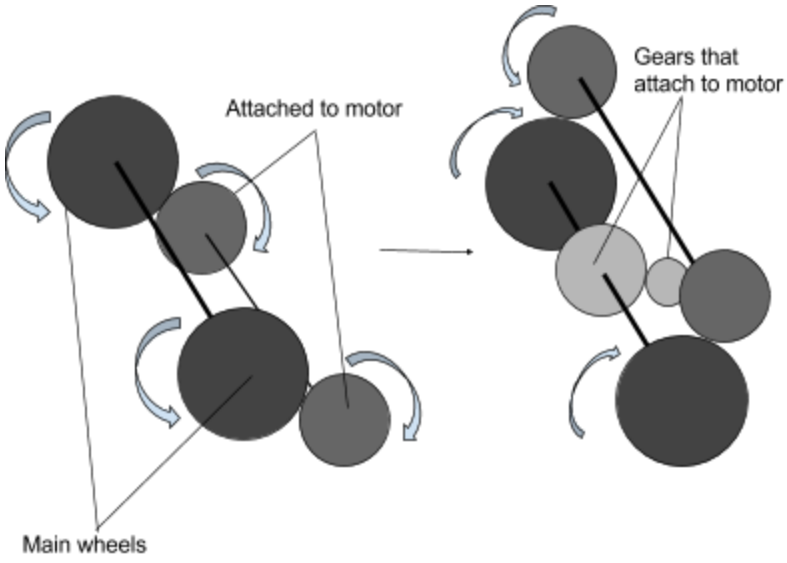
We played around with different methods of translating rotational motion into linear motion. Since we wanted a dotting motion, we needed an up/down motion that would move quickly. We first attempted to incorporate a piston type design, similar to the one shown during lecture, but it was too bulky for our purposes. Since we needed something more streamlined, we used the gear and serrated brick idea that we prototyped in the conveyor belt process. The next challenge was to build something that would hold the pen stable enough after multiple dotting movements. We used the long, slightly curved lego piece to support the pen, and attached

connectors on the sides.. Because the marker was round, it was difficult to surround it with bricks for support, so we used rubber bands stretched across the connectors. Additionally, the friction between the rubber and the marker is great enough to keep it from sliding off.

E. Paper Roller

We introduced a paper roller as a more efficient way of creating the y-direction movement we wanted for the scanner. Initially, we started with just two wheels on the bottom, and found that the paper does not slide because there isn't enough surface contact and friction to move it through the machine. To fix this, we attached a set of wheels on the top of the larger, bottom wheels, and supported it with bricks so they are sitting on top of the wheels with enough pressure. Additionally, the original paper roller used another set of wheels attached to the motor to move the main wheels by surface contact on the bottom. We redesigned it so that only one gear is attached to the motor, and it moves the axle by touching another single gear. This way, the axle is turning both wheels at the same rate instead of introducing the possibility of an error in the two wheel rates due to varying friction on their surfaces (see Diagram 3).

Diagram 3: Paper Roller Design Change



F. Touch Sensor

When we started putting everything together and running our program, we saw that the spacings we told the sensor to stop at and take a reading were not as even as we wished them to be, making it harder to scan an entire pixel design accurately. We incorporated a touch sensor so that the color sensor can return to the same start position every time. It is attached at the end of the track on the right side of our machine, which is where the scanning begins.

Programming:

A. Testing Color Sensor

The first thing we did was run some tests on the color sensor we would be using. We first used the plotSensor example program to understand the syntax for writing the code, but also to determine which of the three modes gave the best accuracy. We tested ColReflect, ColAmbient, and ColColor modes on black and white squares on paper. The ColColor gave the best results, with a clearer dichotomy of black and white since the values, 1 and 6 respectively, are far apart on the spectrum of readings.

B. Scanning

We started by writing a basic script that takes color sensor readings and fills a matrix with those values to display the pattern as an image on the desktop. This helped us with the logic and syntax needed for this functionality. To help improve our accuracy even more, we put in a safety factor by comparing the sensor reading to the value 4. Black is supposed to be the value 1, and white is 6; however, the sensor picks up readings that might be in between. Since our grid is only black and white, to eliminate the intermediate values, in our main code if the reading is less than 4, we will treat the pixel as black, and if it is greater, we will treat it as white.

C. Testing Motor for Conveyor Belt

We decided to use tachometer-based code to make sure the conveyor belt can move as accurately as possible since we are scanning a grid of pixels. One of the large challenges we faced was figuring out how to read the tachometer properly in the program and ensure that it is stopping when we want it to. There were many times when the program would get stuck in a loop, and we learned to display information in certain areas of our code to track where the program is. Later on in our design, we added a “HOME” function in the program with the touch sensor readings in, so that we can make sure the conveyor belt gets back to its origin point everytime it finishes scanning left one row, and right one row. We had to play around with the motor power constants to find an appropriate speed for the scanning motion. It was also difficult to get the motor started sometimes, so we implemented a slight boost every time the motor starts so that the gear overcomes the static friction that keeps the belt in place.

D. Testing Motor for Pen Attachment

We also used tachometer-based code for this functionality. We originally wrote the movement inside the main body of code, but since the same lines of code were repeated 4 times, we decided to write a separate function to make the program cleaner. We started by having the motor simply respond when the color sensor detects a black pixel. Next we moved on to having the marker move downwards, pause for a certain interval, and then move upwards again. We tested different time intervals and motor speeds to find one that works with the marker we have.

E. Integrating All the Movements Together

We started with code for the basic belt movement, going to the left and then right. Then we began to structure our code with comments in places where we wanted to add motor movements or other function calls. We also made sure to have most of the numbers as the values of variables defined at the start of the program. This way, when we needed to adjust values for fine-tuning it became a lot simpler.