Bachelor of Technology: School of Engineering Practice and Technology: Capstone 2

Puzltron System

Group members:

1) **Elliot Wong: 1302982**
2) **Denisan Chandrakumar: 1325986**
3) **Dylan DePodesta: 1327812**

Date: December 10, 2017
Course Code: 4TR3
Core: C02
Supervisor: Dr. Tom Wanyama

Declaration of Authorship:

We, Elliot Wong, Denisan Chandrakumar and Dylan DePodesta certify that this material is our original work. No other person's work has been used throughout this report without citation. We have not made our work available to anyone else.

Abstract:

This is a project report on the Puzltron system. The aim of the system is to create an assembly line capable of shape and pattern differentiation and complete the task it will be given, this project will solve a nine piece jigsaw puzzle, given an image of the completed puzzle. In this report, the overall progress will be provided for the period between the start of TR1 until the end of TR3. Major design concepts, functionality, mathematical models and estimated costs will be discussed in detail, but asides from such, problems and resolution that the team had run into will also be discussed. To better demonstrate the layout and functionality of the Puzltron system, a simulation using the SolidWorks is been created. Dr. Gao has agreed to supply most of the hardware equipment, being the Delta robot, the Dobot and also the conveyor belt, lowering the costs for this project by over 50%. Asides from the hardware, the software portions of the project will also be covered extensively, not only providing in depth technical explanation on the code that was used, sample code will also be provided in the appendices. Lastly, discussion topics like challenges faced and lessons learnt are also included in this report. The project has many applications in assembly plants and can be a learning factory to demonstrate the integration of two robots, visual recognition software and a conveyor belt.

Table Of Contents:

Declaration of Authorship: **1**

Abstract: **2**

Table Of Contents: **3**

Summary of Report: **6**

1. Introduction: **6**

2. Project Background: **7**

2.1 Literature Review: 8

2.1.1 Serial Robot “Dobot”: 8

2.1.2 Raspberry Pi 3: 8

2.1.3 Conveyor Belt: 9

2.1.4 Delta Robot: 9

3. Design Concepts and Project Plan: **9**

3.1 Design Criteria: 9

3.1.1 Process A (Serial Robot): 9

3.1.2 Process B (Image Processing): 10

3.1.3 Process C (Transportation): 10

3.1.4 Process D (Delta Robot): 10

3.1.5 Safety Features: 10

3.1.6 Constraints: 10

3.2 Design Concepts: 11

3.2.1 Serial Robot: 11

3.2.2 Conveyor Belt: 13

3.2.3 Delta Robot: 14

3.3 Economic Analysis: 19

3.4 Project Timeline: 20

3.4.1 Timeline for TR1: 20

3.4.2 Timeline for 4TR3: 21

3.4.2.1 Team 1: Serial Robot and Image Recognition: 21

3.4.2.2 Team 2: Conveyor and Delta Robot: 22

4. Project Implementation: **23**

4.1 Technical Work Completed TR1: 23

4.2 Final Design For 4TR3: 24

4.2.1 Use Case Diagram: 24

4.2.2 Serial Robot: 25
4.2.3 Conveyor Belt: 26
4.2.4 Delta Robot: 27
4.2.5 Changes Made to Initial Design: 28
4.3 Project Development and Group Member Contribution: 28
4.3.1 Dylan DePodesta (Serial Robot): 28
4.3.2 Elliot Wong (Image Recognition): 34
4.3.3 Everyone (Conveyor Belt): 40
4.3.4 Denisan Chandrakumar (Delta Robot): 40
4.4 Testing/Manufacturing of Project: 41
4.4.1 Serial Robot: 41
4.4.2 Image Processing: 43
4.4.3 Conveyor Belt: 44
4.4.4 Delta Robot: 46
4.5 Future Work: 48
4.5.1 Serial Robot: 48
4.5.2 Image Processing: 49
4.5.3 Conveyor Belt: 49
4.5.4 Delta Robot: 49

5. Discussion: 50
5.1 Viability of the Project: 50
5.2 Challenges Faced and Overcame: 50
5.3 Lessons Learned: 52
5.3.1 Serial Robot: 52
5.3.2 Image Processing: 52
5.3.3 Conveyor Belt: 53
5.3.4 Delta Robot: 53
5.4 Practical Applications: 53
5.4.1 Automated Assembly Line: 53
5.4.2 Learning Factory: 54
5.4.3 Silicon Valley Pizzeria: 54
5.5 Problem Solving Methods: 55

6. Conclusion: 55

7. References: 55

8. Appendices: 56
8.1 Appendix A: Table 1: Bill of Materials: 56
8.2 Appendix B: Sample Code: 58

8.2.1 Serial Robot Sample Code: 58
8.2.2 Image Processing Sample Code: 59
8.2.2.1 Contour Finding Code: 59
8.2.2.2 Pattern Recognition Code: 60
8.2.3 Conveyor Belt and Pump Control Sample Code: 61
8.2.4 Delta Robot Sample Code: 63
8.3 Appendix C: Pictures of Components: 64
8.3.1 Serial Robot: 64
8.3.2 Image Recognition: 65
8.3.3 Conveyor Belt: 66
8.3.4 Delta Robot: 67
8.3.5 IR sensor: 68
8.3.6 Dual H-bridge: 68
8.4 Appendix D: Table 2: Expense Report: 69
8.5 Appendix E: Dobot Specifications: 70

Summary of Report:

This technical report will cover the different stages of this project, the design of the system and the implementation. The project report will discuss the following areas:

The **Introduction and Project Background** will discuss the general concept of the project including the scope of the project, the main objectives and the budget for this project. In addition, the reasons why each component was chosen, including pros and cons of each alternative is also explained in this section under the Literature Review.

The **Design Concepts** will discuss the process and flow of the overall project. Also, the safety features and the design constraints that were taken into consideration are included under this section of the report. Each mechanical component has a mathematical model that needs to be worked out and is presented in this section as well. These mathematical models are vital to controlling each of the components after they have been translated. In addition, an economic analysis can also be found in this section for the project design assuming a budget of $500. This section is also where you would be able to find a weekly breakdown of our project plan for the few weeks remaining of 4TR3.

The **Project Implementation** section will discuss the technical details of changes made from TR1 and the current design of TR3. It will also go over progress made for each individual component of the entire system. Whereas the reflection and reasoning for these changes will be explained in a later section. A detailed plan for work that needs to be completed in the future few weeks is also included in this section of the report under Future Works.

The **Discussion** section will present the viability and potential applications of the Puzltron system. The potential applications of the system include automated manufacturing and learning factory. An important part of this section is how the group handled problems that was faced and also new problems that the group has ran into that have not been resolved yet. Reasoning for specific design changes that have been made will also be discussed in this section.

1. Introduction:

The following technical report will discuss the details on the Puzltron Project.

The Puzltron System aims to create a small scale, model assembly line that can complete any given 20-30 piece jigsaw puzzle. This will be done using two robots (a serial robot and a parallel robot) and a conveyor belt in addition to various sensors to keep track of the progress of the process. All system components will be controlled by the two Raspberry Pis interconnected via Ethernet. An entire detailed rundown of the process flow can be found in Section 3.

A box of unsorted puzzle pieces will be scattered in workspace 1, where the camera (Pi Camera, See Appendix C 8.3.2) and pattern detection software will sort through the puzzle pieces and determine the order in which they need to be sent down the conveyor belt. The coordinates for the target piece will be transferred to the serial robot (See Appendix C 8.3.1 for diagram of Dobot) where the piece will be picked up and placed on the conveyor belt (Appendix C 8.3.3) and be transferred towards the next part of the process. When a piece reaches the end of the conveyor belt, it will block the signal of an infrared sensor, causing the conveyor belt to stop running. The puzzle piece, along with every puzzle piece that travels down the conveyor will then become stationary at the same, constant coordinate. The delta robot (See Appendix C 8.3.4) will be ready at this end of the conveyor belt to pick up the piece and place it into the designated spot for the particular piece within workspace 2.

Human input to operate this system is minimal (only scattering the puzzle pieces into workspace 1), the system will automatically complete the task without human intervention.

The Puzltron System can be thought of as a “learning factory”; a concept for a type of learning environment in which theories taught in a classroom environment are brought to life using a small scale representations of the particular system. Because the system is a tangible object, learning objectives can be easily visualized and absorbed by students. For this particular learning factory, concepts of inverse kinematics, image and pattern recognition, and motor control will be studied.

In short, the main objectives for this project are to:

- Program a functional pattern detection algorithm
- Calculate inverse kinematics for delta and serial robots
    - Translate said calculations into smooth, functional code to control robots
- Have suction cup end effector work in harmony with robots to pick up pieces
- Use infrared sensor to switch conveyor belt on and off
- Create a suitable environment for future students to understand these concepts
    - For this very purpose, two types of robots were used in order to broaden the scope of this project.

2. Project Background:

This project is undertaken by three final year B.Tech students to apply knowledge and understanding of robotics (hardware and software) obtained from university and take on a challenging project. The funding for purchasing materials for this project are provided by B.Tech. Each group will be given a set amount of $500 CAD for the completion of the project. The learning objectives of any Capstone project aims to apply project management and logistic skills along with any technical knowledge required. The application of this particular project continues to broaden as society moves away from manual labor and move towards integration

of automation and robotics. For example, manufacturing industries like pharmaceutical companies require color recognition and precise manipulation of small objects (i.e pills and capsules) along a production line. In addition, each project group created a poster for the ISA (International Society of Automation) conference in 4TR1 where the posters were presented to promote the projects of each group.To further promote the project, a simulation of the Puzltron system was presented to competitors and judges.

2.1 Literature Review:

The following section will discuss the reasoning behind the final hardware choices.

2.1.1 Serial Robot “Dobot”:

The “Dobot” is a serial robot that was created by “Shenzhen Yuejiang Technology Co. Ltd” a robotics company in China. The robot itself was designed and created by five engineers who are trying to provide improved, easily accessible industrial robotic arms. The robot was funded by Kickstarter and is now commercially available to the public. Serial robots are used for operations with large workspaces due to its open-loop structure. This is why a serial robot is being used to organize, pick and place the puzzle pieces onto the conveyor belt. The pneumatic pump will also be used on this robot to create a suction that will grip onto the puzzle pieces as they are being moved. This robot will be controlled by a Raspberry Pi and will also use a camera to identify the initial position of the puzzle pieces [2]. The Dobot was chosen to be used for this project because of its suitable size, speed, rotational and pneumatic suction capabilities. Another robot that was considered was the “Lynxmotion robotic arm” it cost roughly the same as the Dobot, (around $500 for the most basic model) and was also compatible with the Raspberry Pi [8]. The reason the Lynxmotion was not chosen in the end was because it did not come with a suction cup gripper and had limited rotational capabilities on its end effector. Furthermore, the Dobot has been provided free of charge by B.Tech and will not incur further costs,greatly influencing the decision-making process.

2.1.2 Raspberry Pi 3:

The Raspberry Pi 3 is the third installment of micro-sized computers from the Raspberry Pi company, it has a 1.2 GHz 64-bit quad-core ARMv8 CPU, with a built-in WiFi and bluetooth modules [5]. It will be used for the visual recognition as well as the motor control on the delta robot and the Dobot. Any form of software (visual recognition as well as robot control modeling equations) will also be ran off of the two Raspberry Pis. The camera is capable of taking HD videos and photos, it connects to the Raspberry Pi via a 15 cm ribbon cable to the CSI port on the Pi. The Raspberry Pi 3 was chosen to control this system because of its visual recognition integration with the Pi camera. An Arduino microcontroller could have been used with a pixy camera, however the camera that integrates with it has lower visual quality [9]. The 1080p resolution of the Raspberry Pi camera allows for much higher accuracy with the visual recognition software. Furthermore, the Raspberry Pi is more powerful compared to the Arduino,

the coding language (Python) is more flexible and easier to work with compared to the C based Arduino language.

2.1.3 Conveyor Belt:

The conveyor belt being used in this project was provided by B.Tech and is 30cm x 10cm. It has a built-in DC motor to control the belt. The frame is made out of metal while the belt is made out of rubber. Other conveyor belts were also being considered because the one that is being used was deemed low quality by one of the supervisors. Prices for a new conveyor belt of similar size ranged from $800-$1000 and was too much for the budget for this project. For that reason, the ideas for finding an alternate conveyor was dropped. The sensors that were considered to be used for controlling the conveyor belt were an ultrasonic sensor or an infrared sensor. An ultrasonic sensor uses sound waves to measure distances, while the infrared sensor uses infrared light to detect the presence of objects. Ultrasonic sensors would not be ideal for this project because the robotic frame of the delta robot would interfere with the readings and an infrared sensor would be more ideal [10]. Also, the height of the puzzle piece is too small (around 3mm in thickness) for an ultrasonic sensor to detect a difference.

2.1.4 Delta Robot:

The delta robot that is going to be used for this project is a prebuilt robot that was designed and constructed by B.Tech graduate students. A delta robot was chosen to put the puzzle pieces into place as they arrive from the conveyor belt due to its high precision, speed and efficiency. In many industries they are used for packaging, pick-and-place, 3D printing, and many more applications. With the help of the visual recognition software and the pneumatic pump suction system, the robot will be able to construct the puzzle at high speeds with minimal error. The motors of the delta robot are controlled with a stepper motor control board connected to a Raspberry Pi 3, the model equation that will be used to calculate the inverse kinematics for operation of the robot can be seen in Section 3.2.3. This specific delta robot was chosen for this task over other delta robots due to its high efficiency and cost effectiveness (it was readily made available to B.Tech students).

3. Design Concepts and Project Plan:

3.1 Design Criteria:

The breakdown of the process is as follows:

3.1.1 Process A (Serial Robot):

The first step of the model assembly line has the serial robot sorting through a pile of scattered puzzle pieces, flipping them over if necessary and also organizing the pieces such that they are

in the desired order of assembly. These pieces are then placed onto the conveyor belt in order, one piece at a time.

3.1.2 Process B (Image Processing):

After ensuring all pieces are right side up, the camera and software will start to analyze each of the pieces and compare them with the completed puzzle in order to look for patterns that are identical. Once it finds a match, it will communicate with the serial robot and instruct it to pick up that particular piece and place it on the conveyor belt. The order of piece placement will be in the order of left to right, top to bottom.

3.1.3 Process C (Transportation):

The transportation phase of the process involves the conveyor belt moving the pieces along to the other robot. The conveyor belt is controlled by a light shield near the end; causing the conveyor belt to stop when a piece reaches the end, where it will be picked up by the delta robot.

3.1.4 Process D (Delta Robot):

The fourth phase of the process has the delta robot pick up the piece at the end of the conveyor belt and assembling the puzzle within its given workspace. Since the puzzle pieces will arrive in the correct order, the delta robot will simply have to place the pieces from left to right and from top to bottom, one piece at a time.

3.1.5 Safety Features:

This system will have some safety features implemented such that should a catastrophic error occur, no injury shall occur. For example, an Emergency Stop button that would activate should the system detect any risk of injury. Aside from this, there will be a form of barrier (software or physical walls) surrounding the assembly line to further prevent risk of injury, if a software barrier is implemented, the system can be programmed to shut down.

3.1.6 Constraints:

Some constraints that arose when analyzing the initial design in *Figure 1*:

- Workspace of the Delta robot is quite limited, which in turn, limits the size of the puzzle that can be used.
- Deciding on the type of sensor that will be used at the end of the conveyor belt that will stop the puzzle piece at the desired location.
    - This issue is that the puzzle pieces are small and thin, causing misreadings to easily occur if an ultrasonic sensor is used.
    - A potential solution as advised from Dr. Zhen Gao is to use an IR sensor, very close to lateral eye level of the conveyor belt. When a puzzle piece comes along and blocks out the signal with its ~2mm height, it will cause

a change in state of the IR sensor IO, however this will only work if the IR sensor is very precise

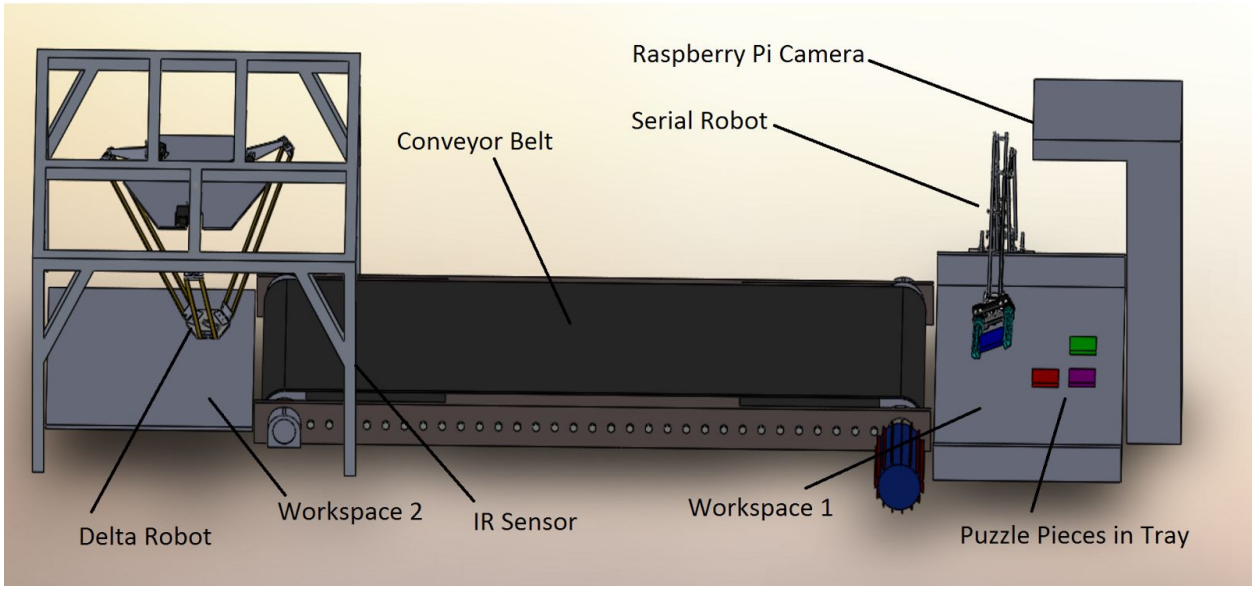


Figure 1: Below shows the initial design for the project

3.2 Design Concepts:

3.2.1 Serial Robot:

The initial process begins with the first camera and the serial robot. This is the location where the puzzle pieces will be sorted and organized. Through image recognition and processing software, the robot will be given an x,y,z coordinate, wherein the robot will then proceed to move its joints in such a way that the end effector will be at the desired, given coordinates. This calculation process is called inverse kinematics. The inverse kinematics model calculations will be shown below.

Looking at *Figure 2* below, the x,y coordinates will be known (after the camera captures an image and communicates the coordinate of the targeted puzzle piece), the lengths of the robotic arm links are known, but angles Θ_1 and Θ_2 need to be found.

$$x = l_1 cos(\Theta_1) + l_2 cos(\Theta_1 + \Theta_2)$$
$$y = l_1 sin(\Theta_1) + l_2 sin(\Theta_1 + \Theta_2)$$

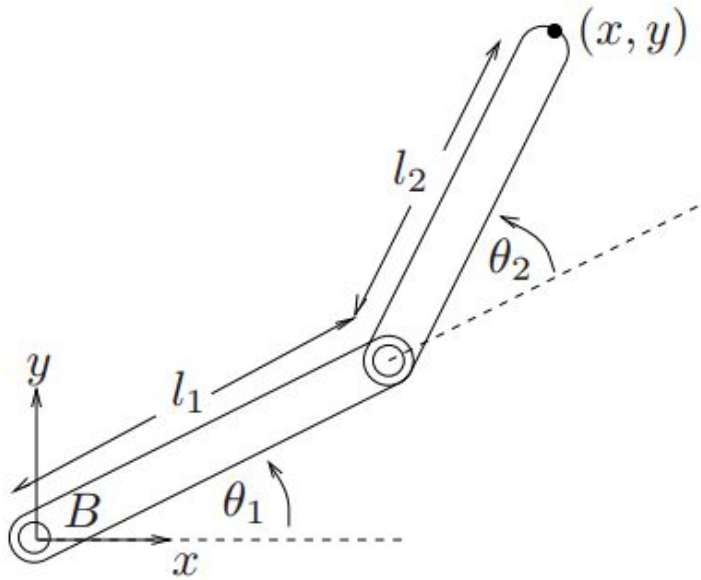


Figure 2: Serial Robotic Arm[4]

Now to develop equations for solving for the joint angles Θ1 and Θ2 given that X and Y are known. To solve this, the Polar Coordinate system(r,ϕ) is needed.

<u>Solving the Θ2 of this system:</u>

By looking at *Figure 3*, it can be determined that the radius $r = \sqrt{x^2 + y^2}$, therefore with the help of *Figure 2*, Θ2 can be solved as follows: [4]

$$\Theta_2 = \pi \pm \alpha \; ; \; where \; \alpha = cos^{-1}\left(\frac{l_1{}^2 + l_2{}^2 - r^2}{2 l_1 l_2}\right)$$

If the value of $\alpha \neq 0$, then there are two values for Θ2, both solutions are correct except one is considered the inverse answer as is indicated by the dotted line on *Figure 3*. Θ1 can now be solved for as: ϕ = Θ1 + β . Θ1 must be solved such that it can yield every possible result for Θ2, those equations can be established as follows: [4]

$$\Theta_1 = atan2(y, x) \pm \beta; \; where \; \beta = cos^{-1}\left(\frac{l_1{}^2 + l_2{}^2 - r^2}{2 l_1 l_2}\right)$$

There are two answers for this equation as well, the answers for Θ1 and Θ2 that matchup correspond with their respective signs (“+” with “+” and “-” with “-” for each of the angles).

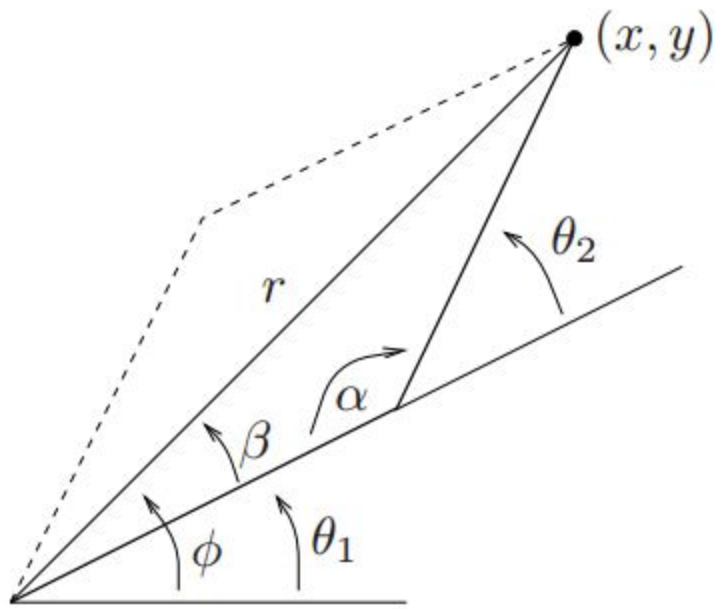


Figure 3: X and Y Position End Effector [4]

3.2.2 Conveyor Belt:

The conveyor belt guides the pieces after they have been selected at the serial robot end of the system towards the delta robot. The pieces will be placed in the center of the conveyor (width-wse) in one end. For this system in particular, the Torque of the motor will be the control variable, the system can be modeled as follows: [1]

$$T = 0.5(D(F + \mu W g))$$

- T = Torque (Nm)
- D = Roller Diameter (m)
- F = External Force (N)
- μ = Coefficient of Friction
- W = Mass of Load (kg)
- g = Gravity = 9.81 m/s^2

Figure 4 shows a model of the conveyor belt system.

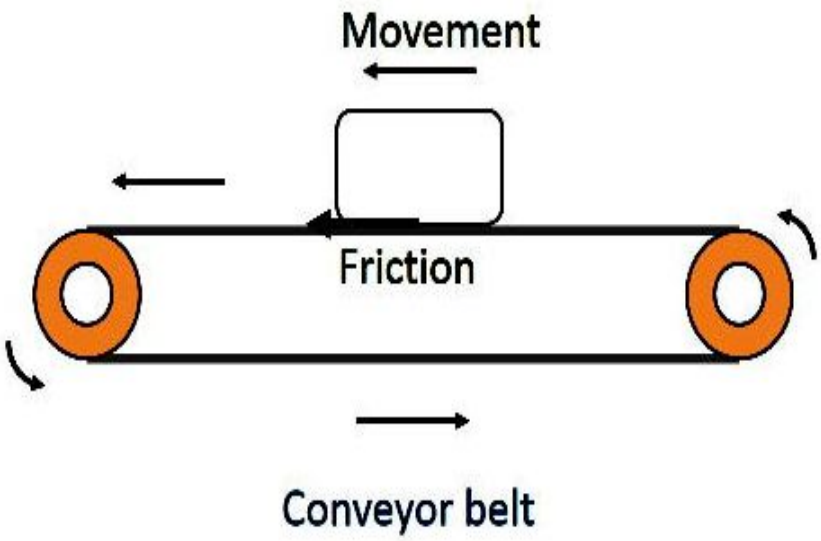


Figure 4: Conveyor Belt Model

3.2.3 Delta Robot:

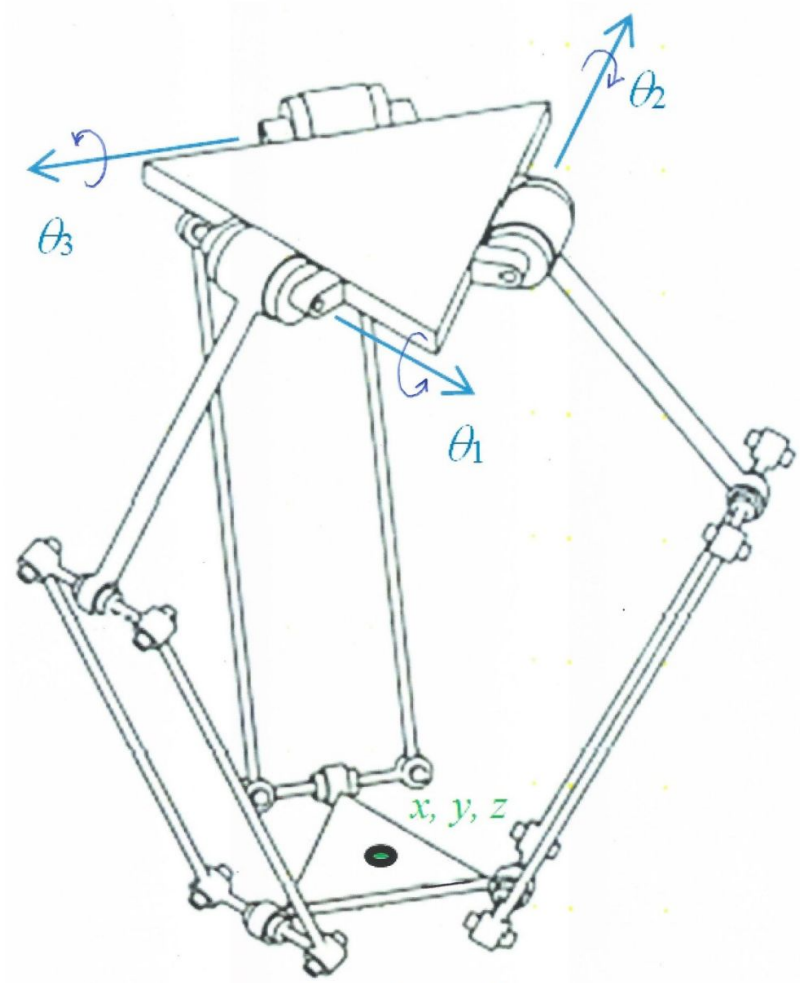
The delta robot in *Figure 5* has 3 Degrees of Freedom (DOF) and has 3 identical RUU (Revolute - Universal - Universal) legs which are determined from the base to the end effector. The underline under the R indicates the joint is an active joint. The control variables of the system are Θ1, Θ2, and Θ3 and are measured from the right hand side.

DOF Analysis:

$$DOF = d(n - g - 1) + \sum_{i=1}^{g} fi$$

- d = order of the system (serial robot = 3 and parallel robot = 6) = 6
- n = number of links + number of frames(Base + End Effector) = (2x3) + 2 = 8
- g = number of joints = (3x3) = 9
- fi = DOF of each joint(RUU in this case) = (3x1) + (2x3) + (2x3) = 15

$$DOF = d(n - g - 1) + \sum_{i=1}^{g} fi = 6(8 - 9 - 1) + 15 = 3$$



Delta Parallel Robot Diagram

adapted from: elmomc.com/capabilities

Figure 5: Delta Robot Angle Rotation [3]

The points B1, B2, and B3 on the base of the robot are considered to be the hip points, points A1, A2, and A3 are considered to be the knees of the robot, and points P1, P2, and P3 are the ankle points of the robot which are connected to the moving platform. L is the length of the joint on the green line on the *Figure 6*. The length of the parallelogram is indicated by l. Both the base and moving platforms are indicated by cartesian coordinates.

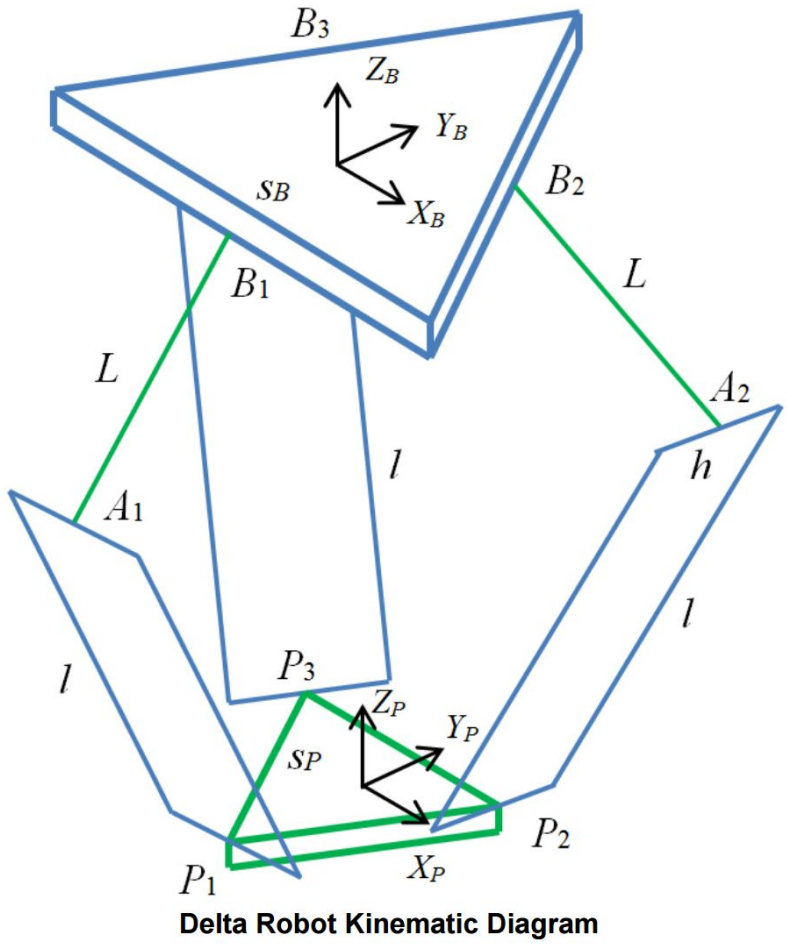
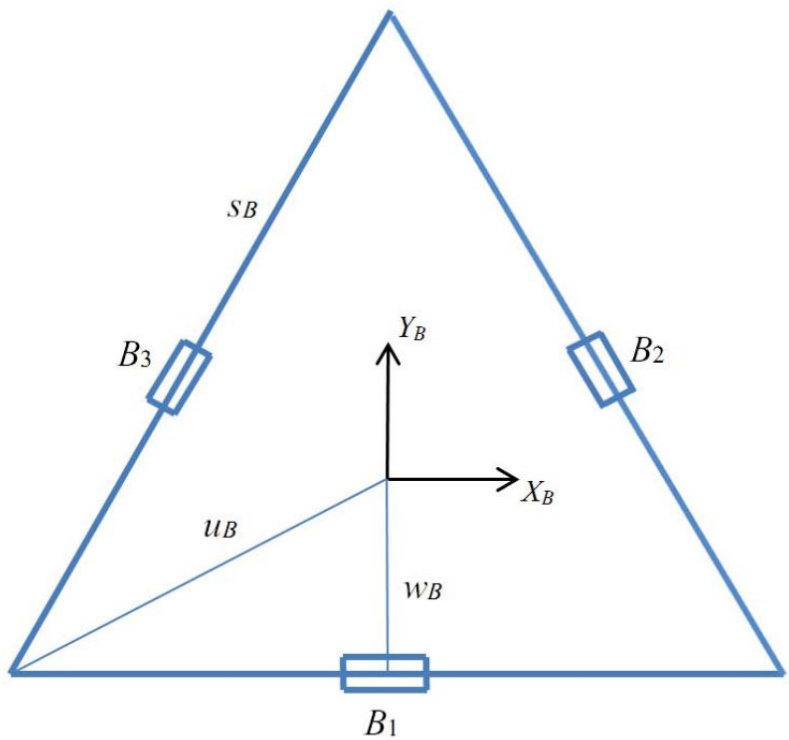
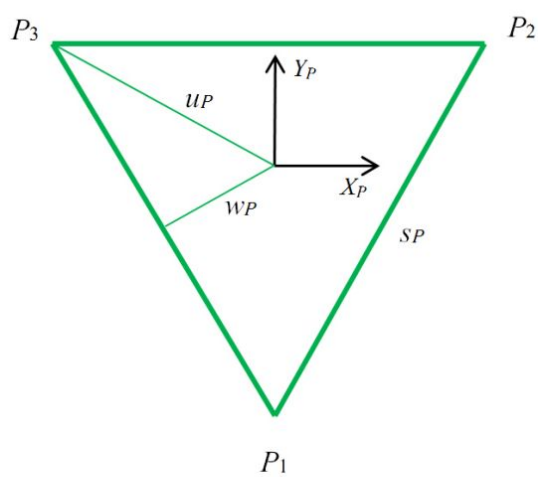


Figure 6: Delta Robot Directional Frame [3]

Figure 7 is about the base and the moving platform and they are situated with an X,Y cartesian plane. The origin of the fixed base is the center of the blue triangle in *Figure 7*. The green triangle has an origin that is also in the center of the triangle.



Delta Robot Fixed Base Details



Delta Robot Moving Platform Details

Figure 7: Delta Robot Base Frame Directionality [3]

The way this problem is going to be looked at is with Inverse Position Kinematics(IPK) which is when the cartesian coordinates are known and the actuated angles need to be found in order for the robot to move around. The revolute points B on the base are all constant points as well

as the points P on the moving platform, they are fixed as well and can be indicated by the following equations: [3]

$$ {}^{B}\mathbf{B}_1 = \begin{Bmatrix} 0 \\ -w_B \\ 0 \end{Bmatrix} \qquad {}^{B}\mathbf{B}_2 = \begin{Bmatrix} \frac{\sqrt{3}}{2} w_B \\ \frac{1}{2} w_B \\ 0 \end{Bmatrix} \qquad {}^{B}\mathbf{B}_3 = \begin{Bmatrix} -\frac{\sqrt{3}}{2} w_B \\ \frac{1}{2} w_B \\ 0 \end{Bmatrix} $$

$$ {}^{P}\mathbf{P}_1 = \begin{Bmatrix} 0 \\ -u_P \\ 0 \end{Bmatrix} \qquad {}^{P}\mathbf{P}_2 = \begin{Bmatrix} \frac{s_P}{2} \\ w_P \\ 0 \end{Bmatrix} \qquad {}^{P}\mathbf{P}_3 = \begin{Bmatrix} -\frac{s_P}{2} \\ w_P \\ 0 \end{Bmatrix} $$

The fixed base equilateral triangle vertices are: [3]

$$ {}^{B}\mathbf{b}_1 = \begin{Bmatrix} \frac{s_B}{2} \\ -w_B \\ 0 \end{Bmatrix} \qquad {}^{B}\mathbf{b}_2 = \begin{Bmatrix} 0 \\ u_B \\ 0 \end{Bmatrix} \qquad {}^{B}\mathbf{b}_3 = \begin{Bmatrix} -\frac{s_B}{2} \\ -w_B \\ 0 \end{Bmatrix} $$

Where:

$$ w_B = \frac{\sqrt{3}}{6} s_B \qquad u_B = \frac{\sqrt{3}}{3} s_B \qquad w_P = \frac{\sqrt{3}}{6} s_P \qquad u_P = \frac{\sqrt{3}}{3} s_P $$

The control point P of the moving platform is given and the angles Θ of the revolute joints need to be calculated. Looking at *Figure 5* the equations of the angles will be calculated analytically as follows:

Three independent scalar IPK equations: [3]

$$ E_i cos\Theta_i + F_i sin\Theta_i + G_i = 0; \; where \; i = 1, 2, 3 $$

When i = 1: [3]

$$ E_1 = 2L(y + a) $$
$$ F_1 = 2zL $$
$$ G_1 = x^2 + y^2 + z^2 + a^2 + L^2 + 2ya - l^2 $$

When i = 2: [3]

$$ E_2 = -L(\sqrt{3}(x + b) + y + c) $$

$$F_2 = 2zL$$
$$G_2 = x^2 + y^2 + z^2 + b^2 + L^2 + 2(xb + yc) - l^2$$

When i = 3: [3]

$$E_3 = L(\sqrt{3}(x-b) - y - c)$$
$$F_3 = 2zL$$
$$G_3 = x^2 + y^2 + z^2 + b^2 + c^2 + L^2 + 2(-xb + yc) - l^2$$

Now that the equations are solved for E, F and G the Tangent Half-Angle Substitution can be applied: [3]

If $t = tan\frac{\Theta}{2}$ then $cos\Theta = \frac{1-t^2}{1+t^2}$ and $sin\Theta = \frac{2t}{1+t^2}$ then substitute into the E, F and G equations.

$$E_i\left(\frac{1-t_i^2}{1+t_i^2}\right) + F_i\left(\frac{2t_i}{1+t_i^2}\right) + G_i = 0 \qquad E_i(1-t_i^2) + F_i(2t_i) + G_i(1+t_i^2) = 0$$

$$(G_i - E_i)t_i^2 + (2F_i)t_i + (G_i + E_i) = 0 \qquad \text{quadratic formula: } t_{i_{1,2}} = \frac{-F_i \pm \sqrt{E_i^2 + F_i^2 - G_i^2}}{G_i - E_i}$$

Solving for Θ, $\Theta_{1,2,3} = 2tan^{-1}(t)$

There are 2 results that come of the quadratic equation and both are correct solutions, one is for a left knee and the other the right knee. [3]

3.3 Economic Analysis:

See Appendix 8.1 for full material list and costs.

Due to the project scope being very straightforward, also due to the major components being provided. There is minimal need for alternate solutions. The budget of $500 is not fully utilized using these estimations.

The calculations for the supplies required, excluding costs of equipment and materials provided by B.Tech:

$108 + $76 + $6 + $13.30 + $40 + $6.54 + $2.61 + $44 + $8 + $76.25 + $30 ≈ $410.7

Estimation of Costs for supplies provided by B.Tech
Delta Robot: ~$3000
Dobot: ~$500

Conveyor Belt: ~$100
Uline Trolley: $163
Total: $3000 + $500 + $100 + 163 = $3763

3.4 Project Timeline:

3.4.1 Timeline for TR1:

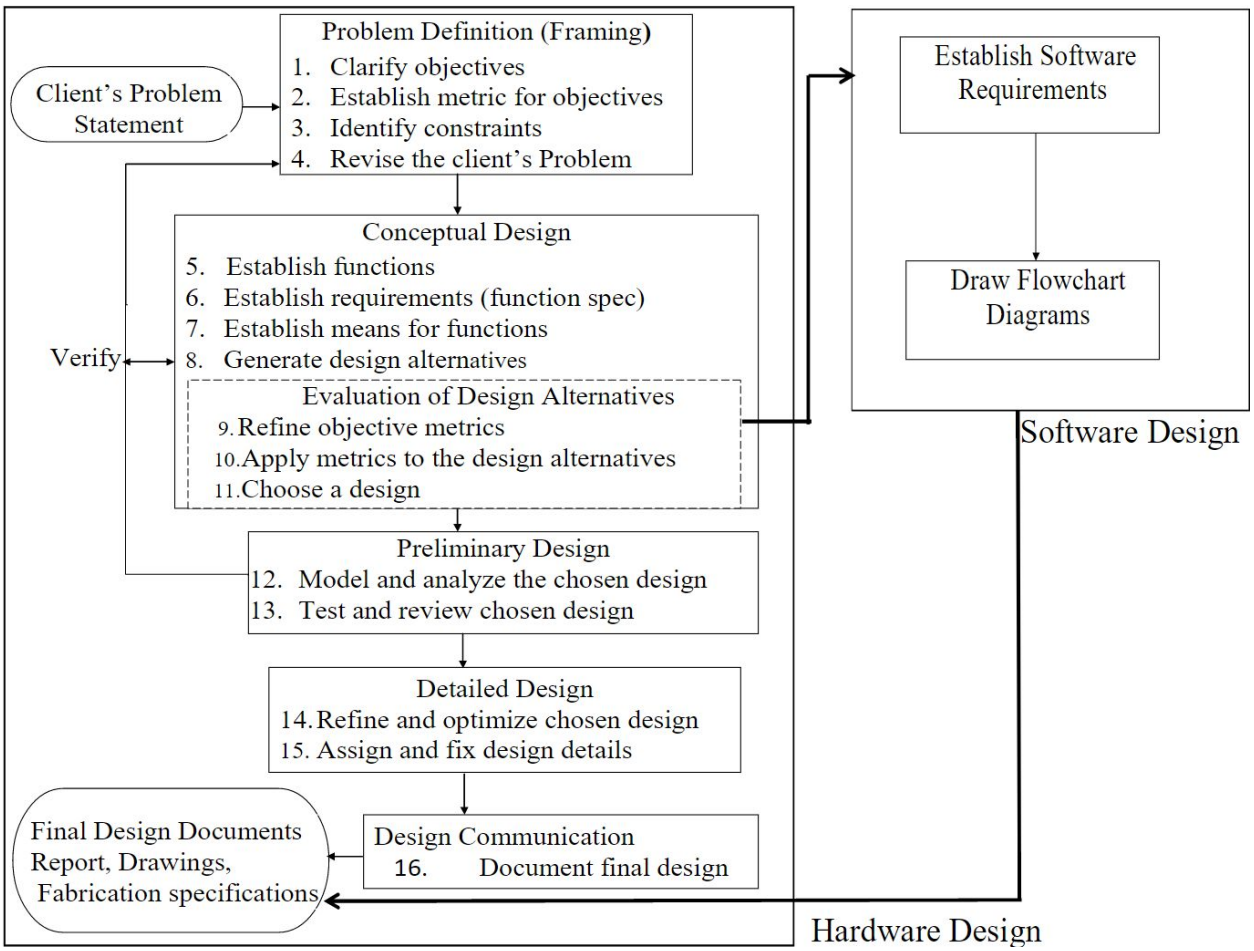


Figure 8: Design Process

January 2017:

- Week 1: Pick TR group members
- Week 2: Purchase log books and begin brainstorming ideas for project
- Week 3: Begin Problem Definition Block (*Figure 8*)
- Week 4: Work on Conceptual Design Block (*Figure 8*)

February 2017:

- Week 1: Begin SolidWorks simulation for the project
- Week 2: Begin working on Midterm Report for TR1

- Week 3: Finalize Midterm Report and work on SolidWorks Simulation
- Week 4: Submit Midterm Report for TR1 and got the Serial Robot, Delta Robot, and the Conveyor Belt

March 2017:

- Week 1: Begin poster for TR1, demonstration of the Delta Robot with Dr. Gao and given working code
- Week 2: Finalize and print poster for ISA Presentation
- Week 3: Went to ISA and presented poster and simulation
- Week 4: Final Presentation for TR1

April 2017:

- Week 1: Handed in Logbooks and worked on Final Report
- Week 2: Handed in Final Report

3.4.2 Timeline for 4TR3:

The remaining parts were purchased over the summer and were delivered on time ready for use for this term, including the IR sensor, cameras, sensors, power supplies, e-Stop, etc..

The project will be split into two teams from this point forward.

Team 1:

- Develop and implement the serial robot
    - Master control system for the Dobot
- Write the image/pattern recognition software using OpenCV

Team 2:

- Work on controlling the motion of the conveyor belt
- Optimize sample code (Appendix B[6]) to create smoother movement
- Implement inverse kinematics to create coordinate system

One group member has left the group, and thus, roles must be assigned with forethought to distribute workload evenly. The tasks for team 2 appears to be simpler and require less work; and so, only one member will be assigned to team 2 while the other two members will be assigned to team 1. An more detailed to-do list for each team is described in the section below, covering benchmarks that should be accomplished on a week-by-week basis.

3.4.2.1 Team 1: Serial Robot and Image Recognition:

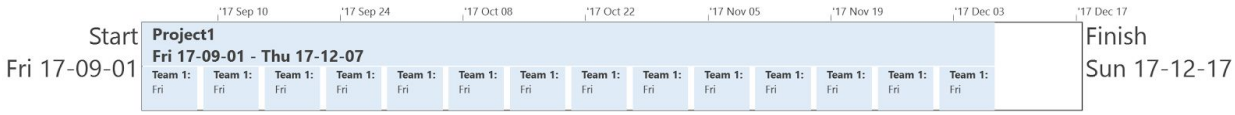


Figure 9: Team 1 Calendar

September 2017:

- Week 1: Start building the Serial Robot
- Week 2: Finish building the Serial Robot
- Week 3: Coding the movement of Serial Robot
- Week 4: Fine tune the Serial Robot movement

October 2017:

- Week 1: Start integrating the Raspberry Pi Camera with the Serial Robot
- Week 2: Familiarize with OpenCV Software Package
- Week 3: Start working towards coding the image recognition program
- Week 4: Continue coding the image recognition software

November 2017:

- Week 1: Get the camera and the software to recognize the puzzle pieces
- Week 2: Integrate the Software with the Serial Robot and debug any errors
- Week 3: Get the Serial Robot to pick up the correct puzzle pieces from the camera
- Week 4: Continue getting the Robot to see the puzzle pieces

December 2017:

- Week 1: Integrating the communication link between the Serial Robot and the Delta Robot and getting the timing of the Conveyor Belt Correct
- Week 2: Final system testing

3.4.2.2 Team 2: Conveyor and Delta Robot:

Figure 10: Team 2 Calendar

September 2017:

- Week 1: Integrate the suction mechanism onto the Delta Robot
- Week 2: Get the existing code that is provided (Appendix B[6]) to work with the Delta Robot and integrate the new Raspberry Pi 3 into the system
- Week 3: Get the ON/OFF suction code to work with the movement of the Delta Robot
- Week 4: Optimize the existing Delta Robot Code to run a lot smoother

October 2017:

- Week 1: Troubleshoot any errors that might occur in the process
- Week 2: Get the IR sensor working with the Conveyor Belt and have pieces stop at the designated pickup location
- Week 3: Integrate the IR Sensor with the Delta Robot

- Week 4: Get the Delta Robot to pick up the pieces from the Conveyor Belt

November 2017:
- Week 1: Troubleshoot any errors that the Pick command might have with the pieces
- Week 2: Code the placement area based on the amount of puzzle pieces that are known
- Week 3: Test the pick and placement algorithm and ensure consistent results
- Week 4: Troubleshoot remaining problems

December 2017:
- Week 1: Integrating the communication link between the Serial Robot and the Delta Robot and getting the timing of the Conveyor Belt Correct
- Week 2: Final system testing

4. Project Implementation:

4.1 Technical Work Completed TR1:

Most of the work that has been done in TR1 focused on figuring out the logistics and workflow of the system, along with acquiring most of the hardware (ie. the two robots and the conveyor belt). A lot of time and effort went into the SolidWorks simulated demonstration, which was presented at the annual International Society of Automation (ISA) Exhibition along with the project poster.

Recap of Progress from TR1

Serial Robot	A spare Dobot has been provided by Dr Gao, all components are present. The control method and programming have not been provided and will have to be developed.
Conveyor Belt	Although a conveyor belt was provided for free. It was deemed rather clunky and so alternatives were searched for online. The two that were found were respectively $850 and $900. These costs were astronomical for its mere purpose of transporting puzzle pieces. Not willing to sacrifice cost efficiency for a small objective, it was decided to forego these alternatives.
Delta Robot	A Delta Robot built by graduated B.Tech students was also provided to us by Dr. Gao. After running functionality tests, it was decided that this robot would be suitable for this project. Although, some sample code was provided, it was minimal and needed to be optimized.

4.2 Final Design For 4TR3:

4.2.1 Use Case Diagram:

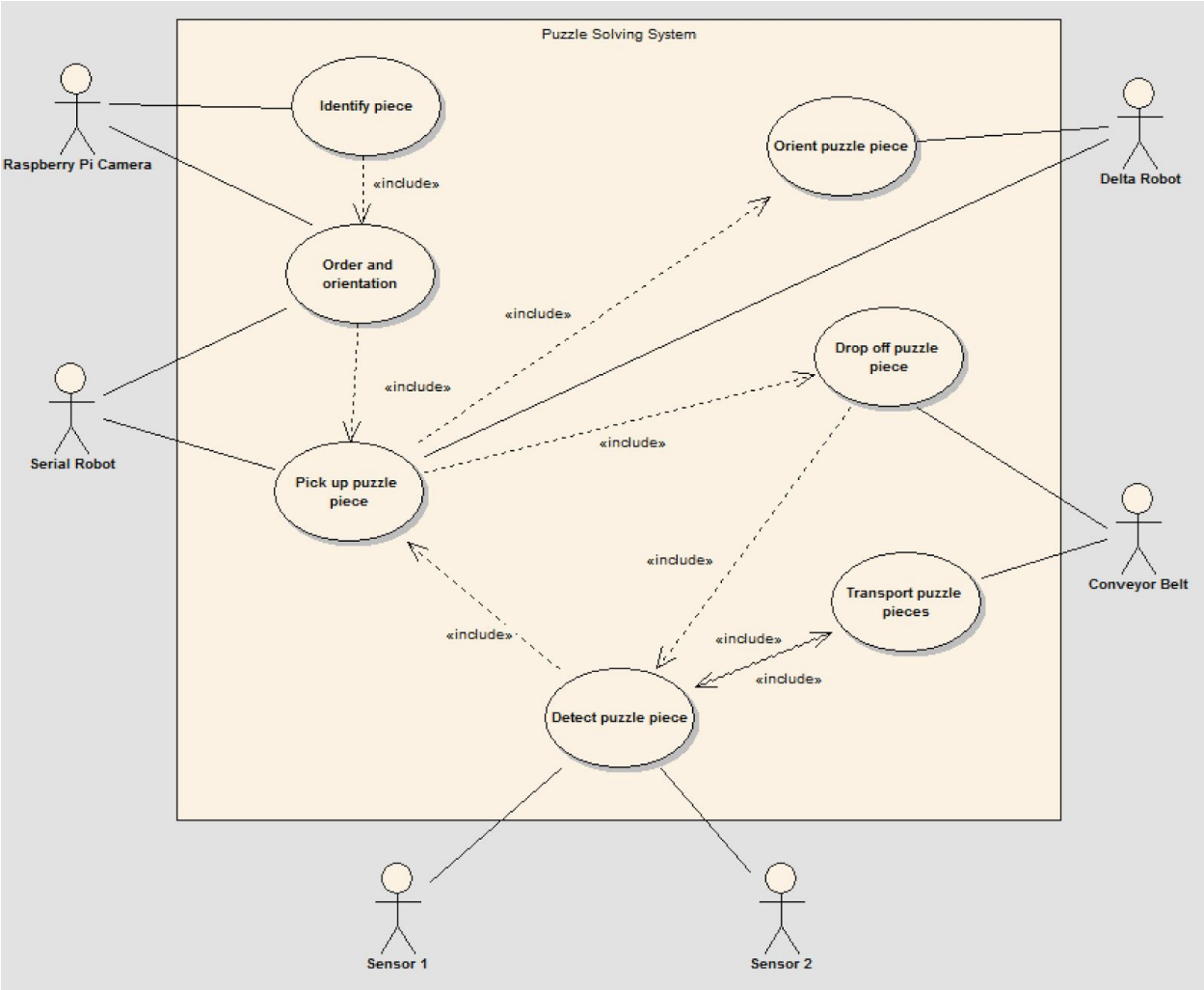


Figure 11: Use Case Diagram

4.2.2 Serial Robot:

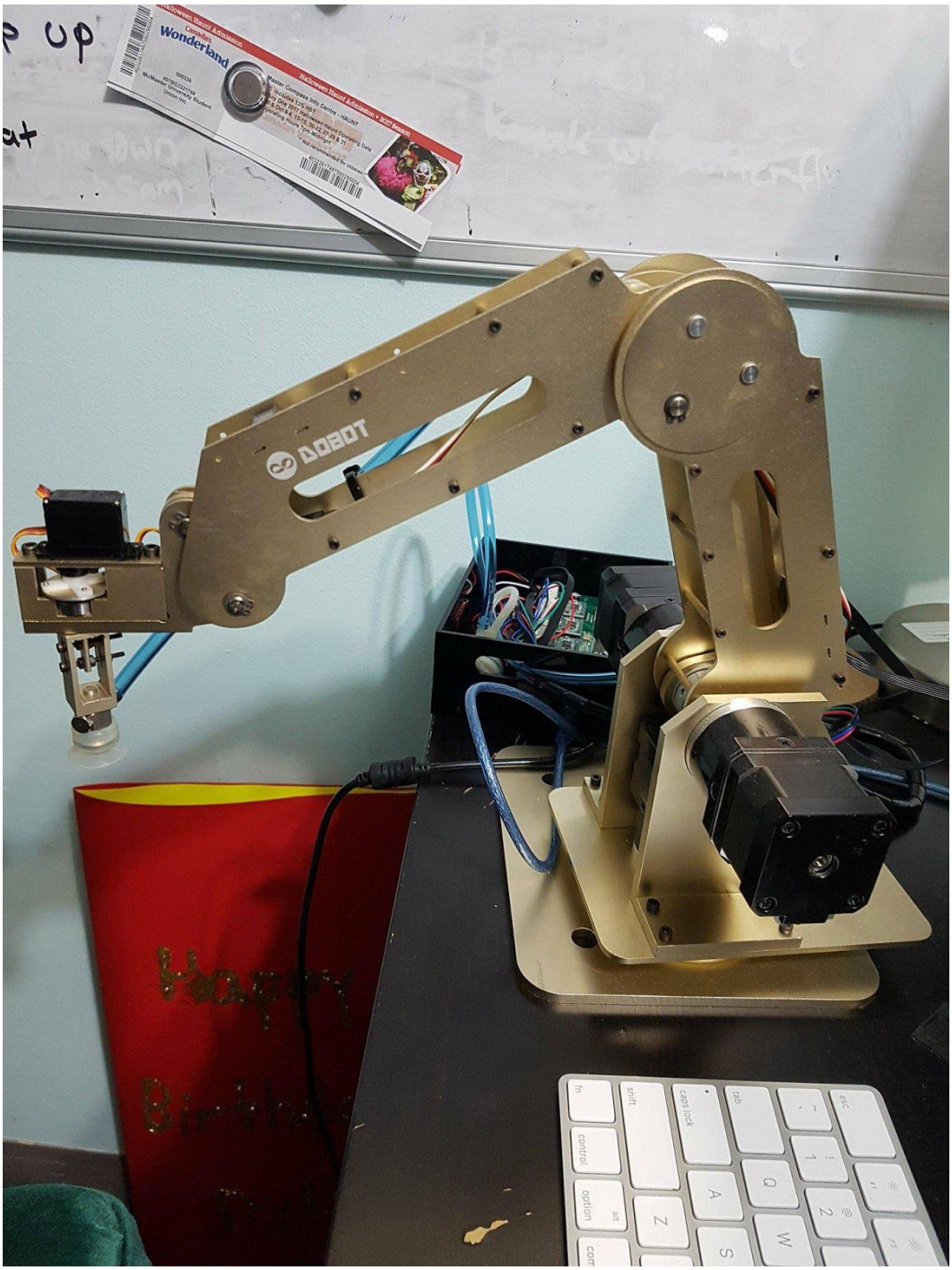


Figure 12: Final Design of Serial Robot

This is the final design of the Dobot that will be implemented into the Puzltron system. It is going to be used to pick up, rotate, and flip the puzzle pieces onto the conveyor belt in the correct

position according to the visual recognition software. Afterwards, the Raspberry Pi camera takes a picture of the remaining puzzle pieces to determine the next piece that needs to be moved.

In the final design, the claw-style end effector of the robot has been changed to a pneumatic suction cup end effector due to it being a simpler means to accomplish the same goal.

4.2.3 Conveyor Belt:

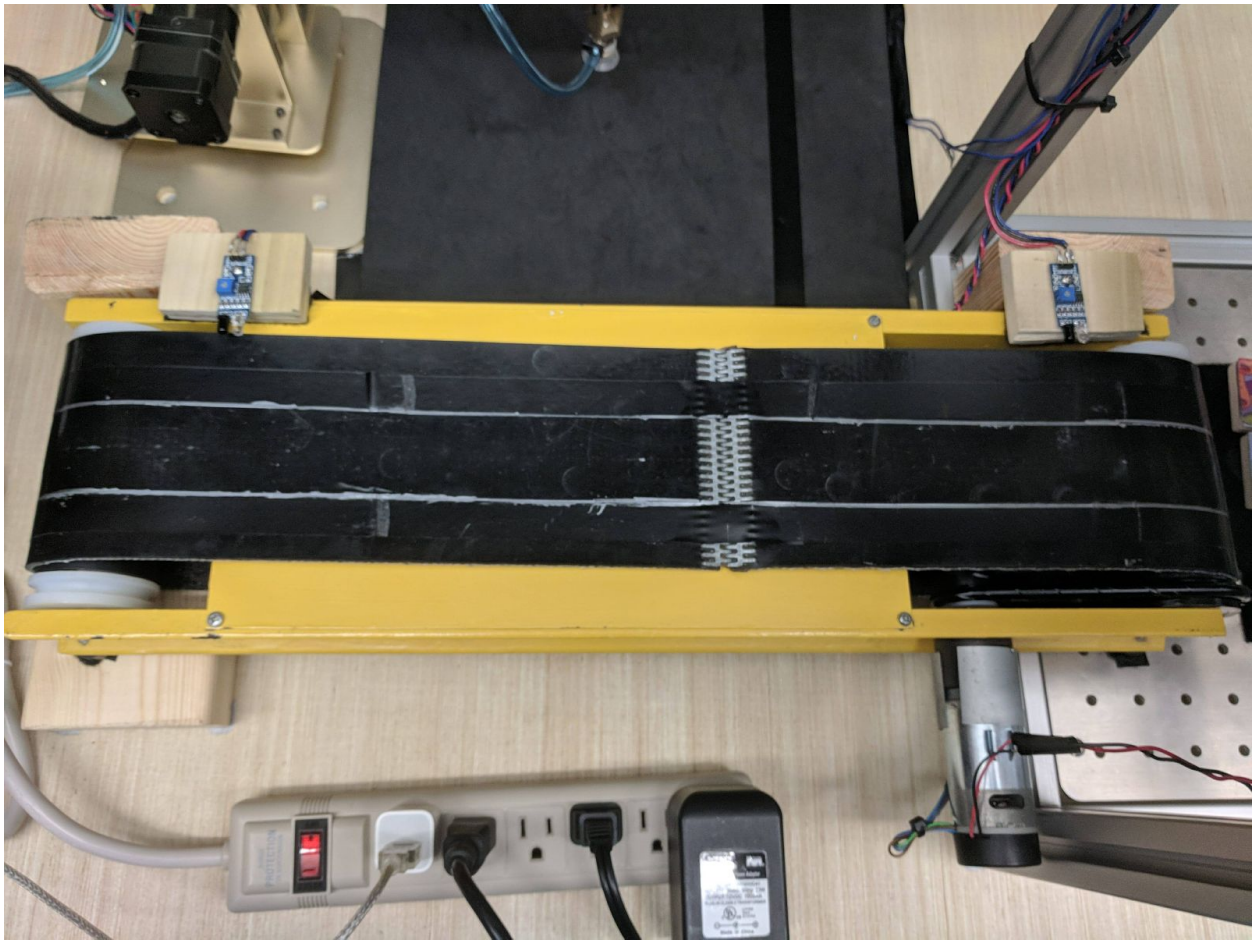


Figure 13: Final Design of Conveyor Belt

This is the final design of the conveyor belt that is going to be implemented into the Puzltron system. It is going to be used to transport puzzle pieces from the Dobot to the delta robot. The conveyor belt is driven by a DC motor, which will be connected to an Arduino connected to the Raspberry Pi due to insufficient I/O pins (all I/O pins are used up by the delta robot).

The serial robot will be programmed to place each puzzle piece in the middle of ends of the conveyor before it starts moving down to the delta robot.

4.2.4 Delta Robot:

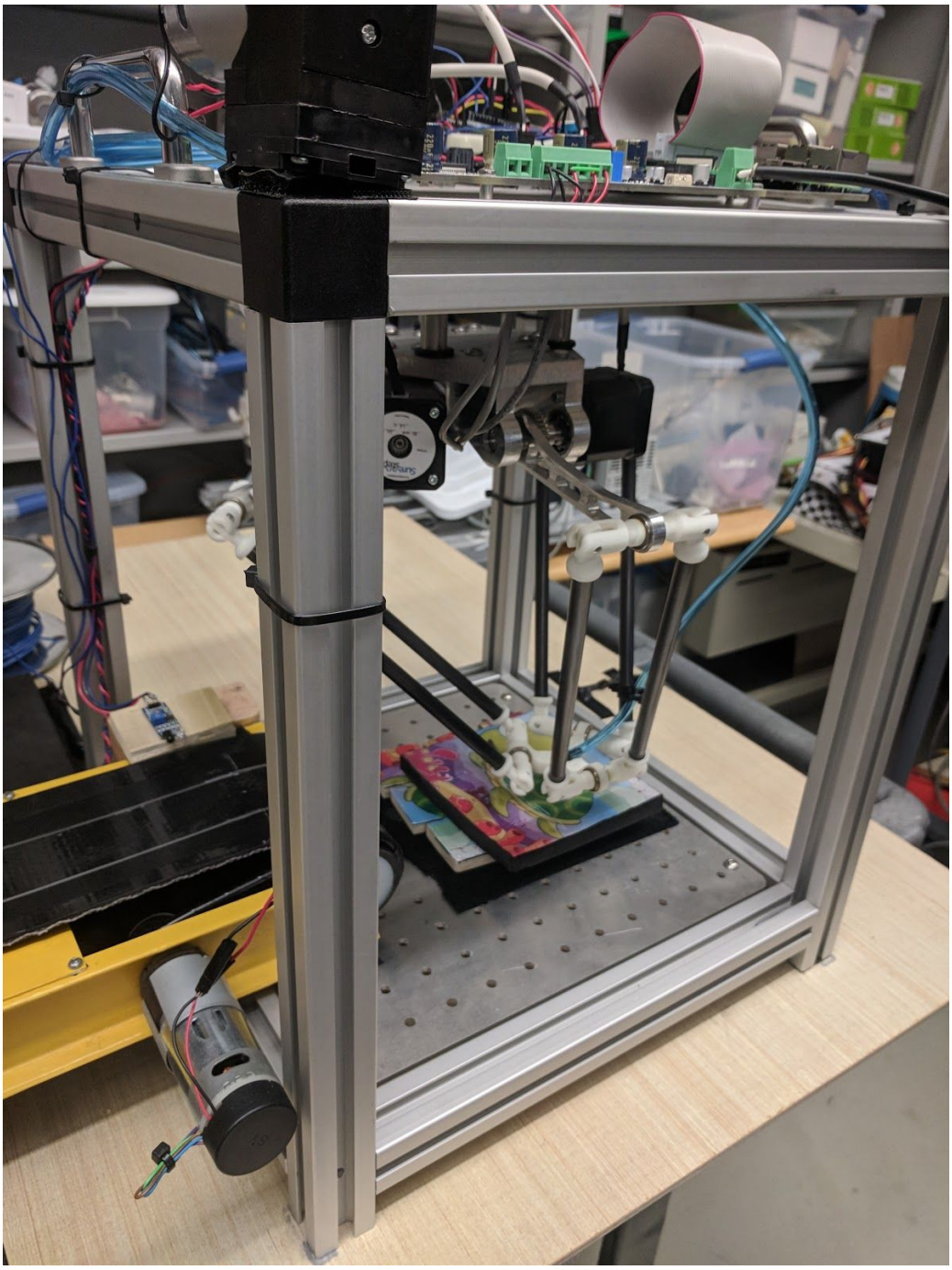


Figure 14: Final Design of the Delta Robot

This is the final design of the delta robot that is going to be implemented into the Puzltron system. The delta robot will pick up the puzzle piece from the conveyor belt once it stops at the

IR sensor location. Through the software and mathematical calculations done by the program, the delta robot will pick and place the pieces into the proper locations to put the puzzle together.

4.2.5 Changes Made to Initial Design:

The design for the system was reconsidered many times due to cost-limitations and space. The first design consisted of the two Pi cameras, a serial robot, a delta robot and conveyor belt. Due to the poor positioning of the camera on the delta robot, the decision was made to remove the camera on the delta robot and keeping one camera for the serial robot. Another design change made was the positioning of the conveyor belt so it is closer to the platform of the delta robot. This change allows the delta robot to reach over the end of the conveyor robot and grab the puzzle piece off the conveyor belt. Before this change, the design of the system would have the puzzle piece fall onto workspace 2 after its ride on the conveyor belt; this change would cause inaccuracies due to slightly different landing spots.

As previously mentioned, the claw-style end effector was also changed in favor of a suction cup system due to higher simplicity.

4.3 Project Development and Group Member Contribution:

4.3.1 Dylan DePodesta (Serial Robot):

It was discovered that a Arduino Mega 2560 was being used to control the motor control board as well as the pump controls, there is an external board that is attached to the Arduino Mega 2560 that uses all the pins. Getting the Dobot to work with the Raspberry Pi included the following: The motor controls and motor drivers are on the FPGA board, so the Arduino Mega 2560 sends queued commands to the FPGA board. The FPGA board must be flashed with a configuration HEX file before the FPGA board can receive those queued commands from the Arduino Mega 2560. Using OpenDobotProject from GitHub, there was a premade configuration file that was used to flash the FPGA board. Using AVRdude command line program, the HEX was loaded onto the Arduino Mega 2560, which was sent to the FPGA board using SPI interface. A constraint that came up was that the FPGA board needed to be cleared before uploading a newer version onto it. From the same library, there was a sample driver code that was used to test out the robot, which worked.

The robotic arm is able to provide suction and release the puzzle pieces. There were several modification that the puzzle pieces had to undergo so that systems would function properly. First, the puzzle pieces had to be raised to account for the deviation in sensor height on the conveyor belt, the IR sensors would not be able to detect the very thin puzzle pieces so they had to be raised in order for detection to occur. Second, the puzzle pieces needed velcro underneath them so that they would stick to the reciprocal velcro piece located on the Delta Robot platform. Finally, since each piece had a lot of glare and was hard for the Image

Processing system to work properly each piece was fitted with a photocopied piece so there was no glare.

Figure 15: Puzzle Piece

The pump control on the Dobot is controlled via a series of drivers that have been called in the main code set. The drivers that are called from another file are called “PumpOn” and “ValveOn”. The code in *Figure 16* displays a function that can be called when the pump needs to be turned on and off.

```
def repeatUntilQueued(on):
    ret = (0,0)
    while not ret[0] or not ret[1]:
        ret = dobot.PumpOn(on)
    ret = (0,0)
    while not ret[0] or not ret[1]:
        ret = dobot.ValveOn(on)
```

Figure 16: Pump Control code

The inverse kinematics has been coded into the robot, but it still under maintenance and some debugging is required in order to get the robot working with this code. The inverse kinematics

code works for the most part and there are some bugs within the code. The inverse kinematics that has been programmed works by taking the current position of the arm and subtracting off the location you want to go to and it will travel the difference in x, y, and z.

$$CurrentX - NewX = MoveInX$$
$$CurrentY - NewY = MoveInY$$
$$CurrentZ - NewZ = MoveInZ$$

The serial robot runs off of 3 coordinate systems, there is World, Tool, and Base *Figure 17* below. The world coordinate system uses a millimeter scale for the robotic arm to move around with. The base coordinate system also runs off of a millimeter coordinate system but it is a millimeter rotation as opposed to a linear translation like the world coordinate system. The base coordinate system has a 270° of rotation.The Tool coordinate system is the end effector of the robotic arm and that was a servo motor that was connected to a suction system. The end effector was 50.9 mm in length and that needed to be taken into consideration when finding the z - coordinate in the world coordinate system.

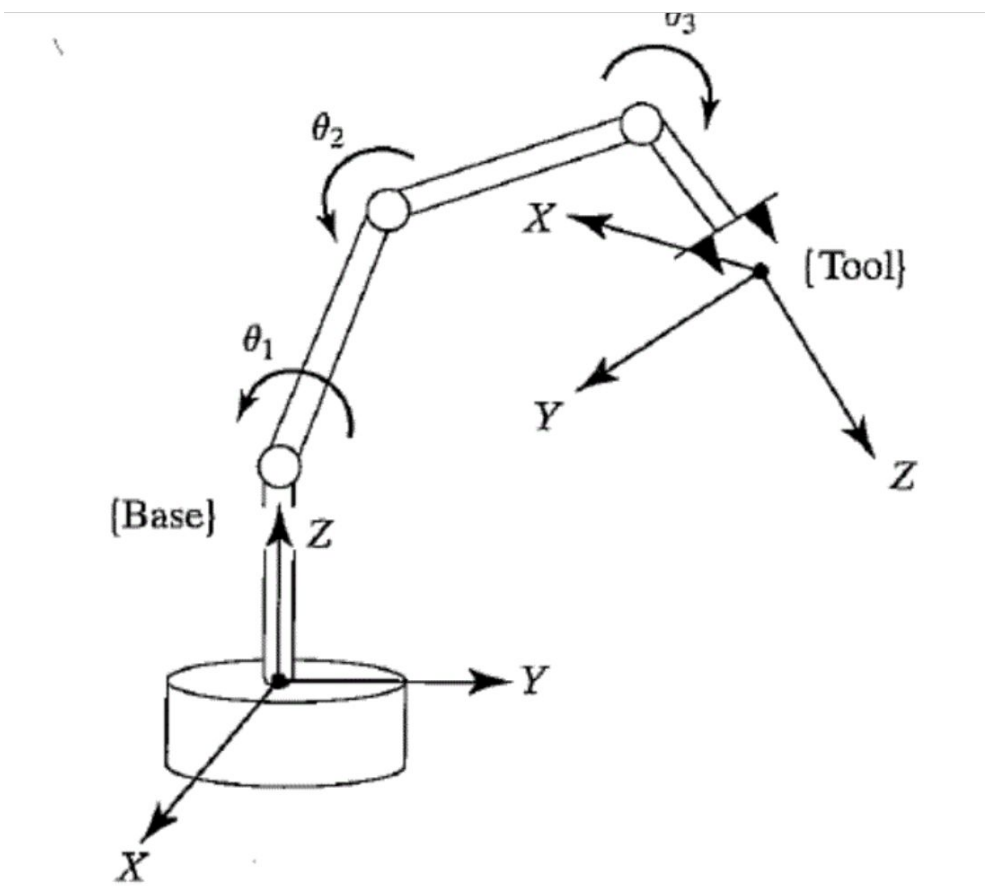
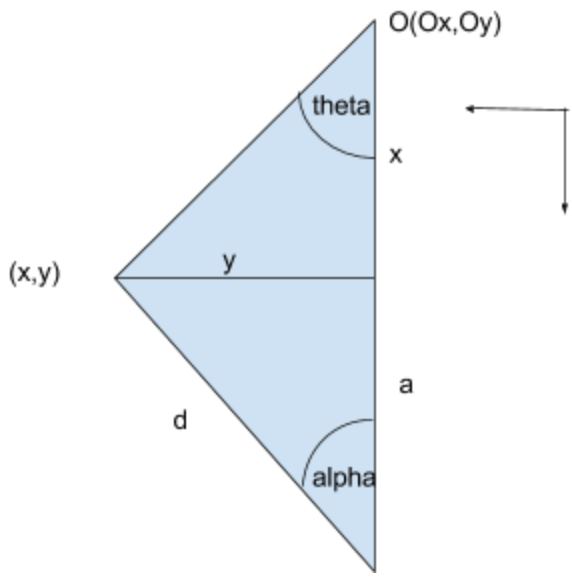


Figure 17: Robot Coordinate Systems (World, Base, and Tool)

The camera is integrated into the raspberry pi,, it is able to take photos and videos and save them to any desired directory. Analysis of these photos will be done via OpenCV in python. The camera has its own coordinate system that needs to be integrated into the serial robots. The camera coordinate system is a pixel coordinate system which needs to be converted to the millimeter world coordinate system of the serial robot. The pixel coordinate system starts at the top left hand corner of the picture area and positive y is going down and positive x is to the right and this (0,0). As for the world coordinate system on the serial robot the farthest point away from the robot and on the ground is considered (0,0,0) and this point is constrained *Figure 18* below. In *Figure 19* below you can see how the 2 coordinate systems are layed out. There needs to be a conversion between then and using some math that conversion can be achieved as follows:

$$(Ox - Px) = Cx$$
$$(Py - Oy) = Cy$$



$$dCos(\alpha) = a - x$$
$$d = \frac{(a-x)}{Cos(\alpha)}$$
$$d^2 = y^2 + (a - x)^2$$
$$\alpha = arcCos(\frac{(a-x)}{d})$$

This is how the camera's pixel coordinate system will convert to the serial robots coordinate system.

Figure 18: Point (0,0,0) which is constrained

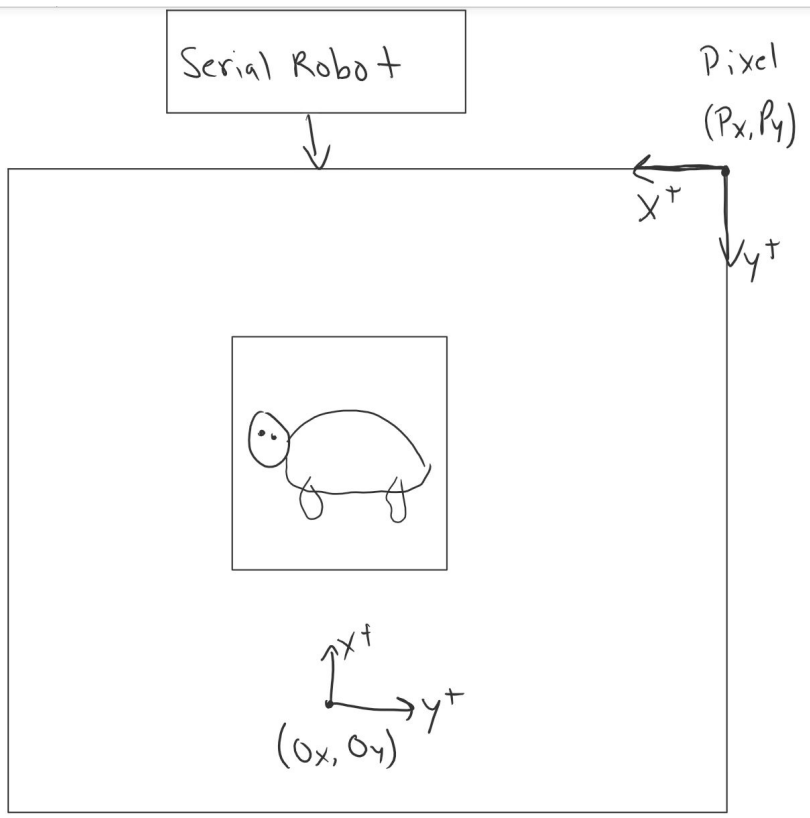


Figure 19: Pixel and Serial Robot Coordinate Systems

The end effector of the serial robot needed to rotate to account for the orientation of the puzzle piece on the platform. To account for this a mathematical model was used:

Looking at the “S” graph in *Figure 20* below it can be see that a shape is parallel to the x and y axis and there is going to be some rotational analysis that is applied to each of the coordinates (x1,y1), (x2,y2), (x3,y3), and (x4,y4). Since the the coordinates are known for both graphs “S” and “S’” all that leaves is to find the angle so that the end effector of the serial robot can apply that angle of rotation. In order to find the angle of rotation you can apply the following mathematical steps:

1) Pick any point on the S plot, so for the example in *Figure 20* below, the point (x1,y1) will be chosen off the S plot.

$$r = \sqrt{x1^2 + y1^2}$$

2) The value of r that was calculated from the S plot should be the same as r from the S’ for the same point (x1,y1).

$$r = \sqrt{x1^2 + y1^2}$$

3) In order to find the angle of rotation between S and S’ can be determined as follows.

$$S: \theta_1 = tan^{-1}(\frac{y1}{x1})$$
$$S': \theta_2 = tan^{-1}(\frac{y1}{x1})$$
$\theta = \theta_2 - \theta_1$; where θ is the angle of rotation between the plots S and S’ in *Figure 20*

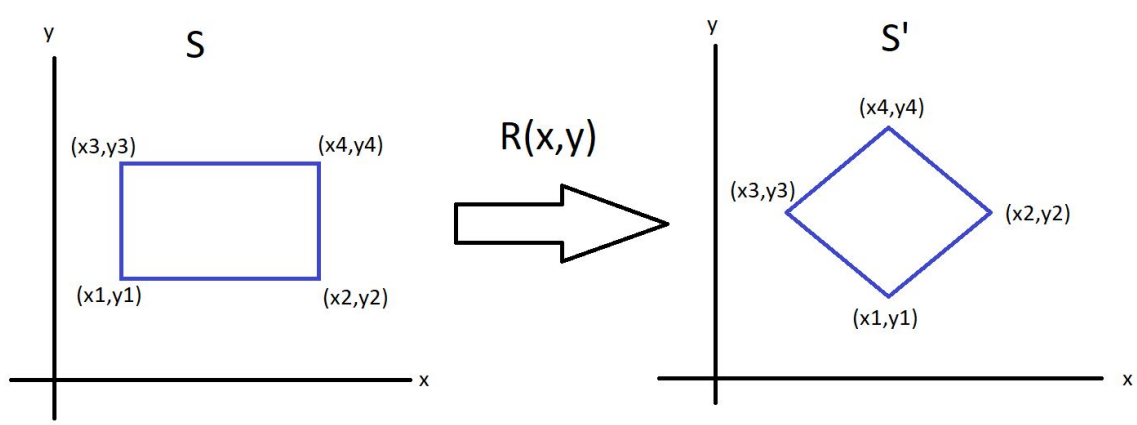


Figure 20: Rotational movement

4.3.2 Elliot Wong (Image Recognition):

The image recognition portion of the project is the software backbone of the project. Its purpose is to identify puzzle pieces and match them up to the full image of the puzzle. A lot of time was spent on installing the plugin, OpenCV that allows python to perform image analysis, this is because asides from the regular OpenCV 3.3 library, an additional package outside of the default package called “xfeatures2d” was needed. Cmake files were needed to be made from these two components then built and compiled.

The image recognition portion of the project is the software backbone of the project. Its purpose is to identify puzzle pieces that will be laid out on the workspace. These puzzle pieces will be matched up with the full image of the puzzle, and their respective locations will be determined. These coordinates will be directly sent to the serial robot in an array as a form of instructions. Below, the step by step procedure of this analysis process will be described.

The image recognition will be completed using OpenCV 3.3-dev on Python 3.

<u>Taking the Photos:</u>

The Pi Camera can be controlled through a free library called “picamera”. The photos will be captured and outputted to JPG images to a designated folder through the command:

```
with picamera.PiCamera() as camera:
      camera.capture('pieces.jpg')
```

However, before the photo is taken, the camera must be set up properly with the commands:

```
camera = PiCamera()
camera.resolution = (1024, 1024)
camera.ISO = 100
camera.brightness = 70
Camera.contrast = 70
camera.start_preview()
```

As the puzzle and the work space are both square, it was appropriate that the image resolution is also square. The ISO is also set to a much higher value such that the image quality can be improved. A lighting contraption (made of LED strips) was added to the system such that increasing the shutter speed would not darken the image completely. The brightness of the output image was also increased in order to assist in this same regard. The contrast was increased in order to retain the sharp color of the image before the LED strips washed out the image.

Using these settings, two photos are taken, an image of the full puzzle, and then an image of the puzzle pieces scattered about the workspace.

Finding the Contours:

The first step in the image analysis is to identify the size and contours of both pictures. This way, only contours of the desired size range can be remained and in turn, analyzed.

The first step of this is to convert the image into a grayscale using the command:

```
image = cv2.cvtColor(image, COLOR_RGB2GRA)
```

Next, the image must be blurred such that the pixel differences transition smoothly. A gaussian blur mask was used.

```
blurred = cv2.GaussianBlur(image, (5,5), 0)
```

This blurred image will then be put on a binary threshold in order to change the values of the pixel values. The values will be altered such that values below this threshold will be changed to 0 (black), while values above the threshold will be changed to 255 (white).

```
threshold = cv2.threshold(blurred, 40, 255, cv2.THRESH_BINARY)[1]
#where 40 is the threshold
```

Results can be seen below:

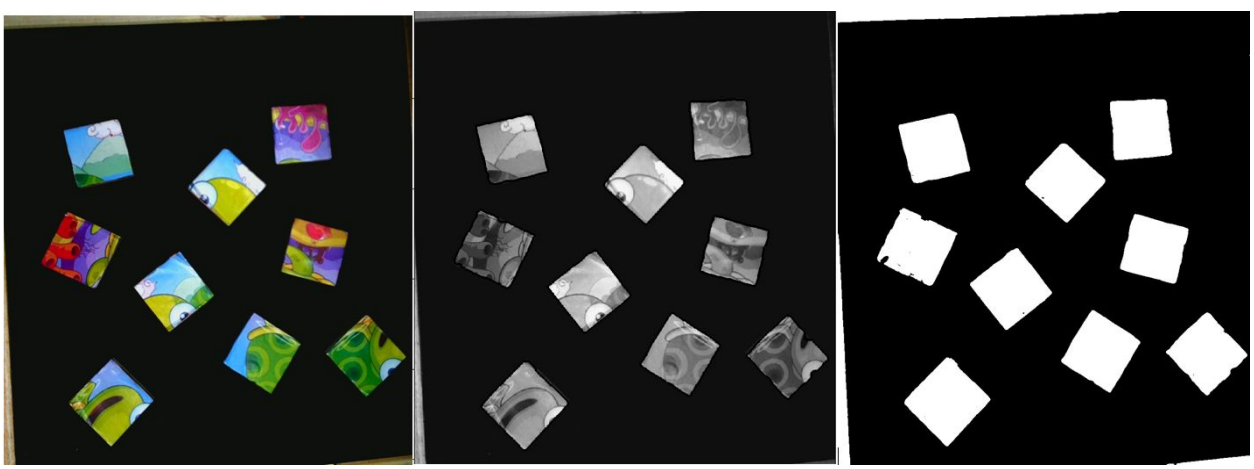


Figure 21: Original, Grayscale, Thresholding

The contours can be found using:

```
_,contours,_=cv2.findContours(threshold,cv2.RETR_LIST,cv2.CHAIN_APPROX_NONE)
```

This function returns information about the contours found in the image. The most important information that is needed are the moments and the area of each contour.

The contours must firstly be filtered based on size. Any contours with areas that are outside the predefined values are eliminated from the array or contours. Only these contours will be returned to the main function.

The moments of each individual contour can be analyzed in order to find the center points of each piece:

```
M = cv2.moments(contour)
cx = int(M['m10'] / M['m00'])
centerx.append(cx)
cy = int(M['m01'] / M['m00'])
centery.append(cy)
#the center points are accumulated into arrays so they can be gathered all at
once
```

Each piece is then cropped out and saved into a separate image file within the code. This can be done based on the center points and areas previously found. Each of the pieces will be analyzed and compared to the full picture separately. Firstly they will be put through the pattern detection algorithm.

<u>Pattern Detection:</u>

The grayscale image is now passed through the SIFT pattern detection algorithm.

SIFT is an algorithm developed and published in the University of British Columbia used to detect and describe local features in the field of computer vision.

It is approximated by the mathematical equation below:

Scalar Transform

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

L = Scalar Transform
G = A variable-scale Gaussian
I = Input image

Keypoint location can be found using the Difference of Gaussian method:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$$

The scalar transform maxima subtracting the scalar transform minima.

Firstly, the algorithm identifies *corner points* within the picture by identifying sharp changes in pixel values. As seen in *Figure 22*.

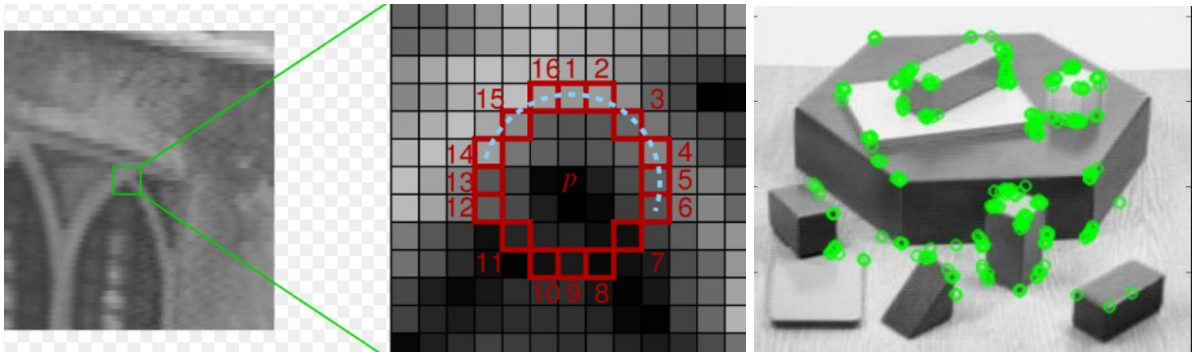


Figure 22: Left and Middle: Corner points finding method
Right: Corner points found

Through the SIFT algorithm, each pixel is compared with its neighboring pixels, the gradient direction and intensity is compared and turned into a histogram (As seen in *Figure 22*).

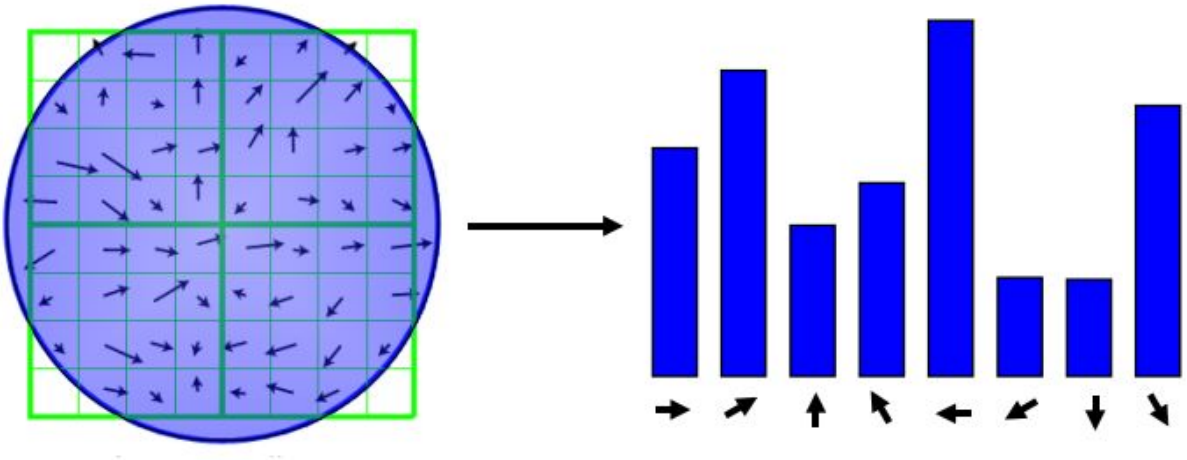


Figure 23: Histograms made from comparing

Pairs of histograms that are found to be similar will be identified as a pair of descriptors that are matching points, based on a given distance value.

If both a corner point and a matched pair of descriptors was found in the same patch of pixels. A match will be identified between the two. Each match is returned as a keypoint variable put into an array. From this, the coordinates of each match are extracted.

Piece Classification:

With the two sets of coordinate arrays, a series of comparison statements were made according to the coordinate value on the “full” image.

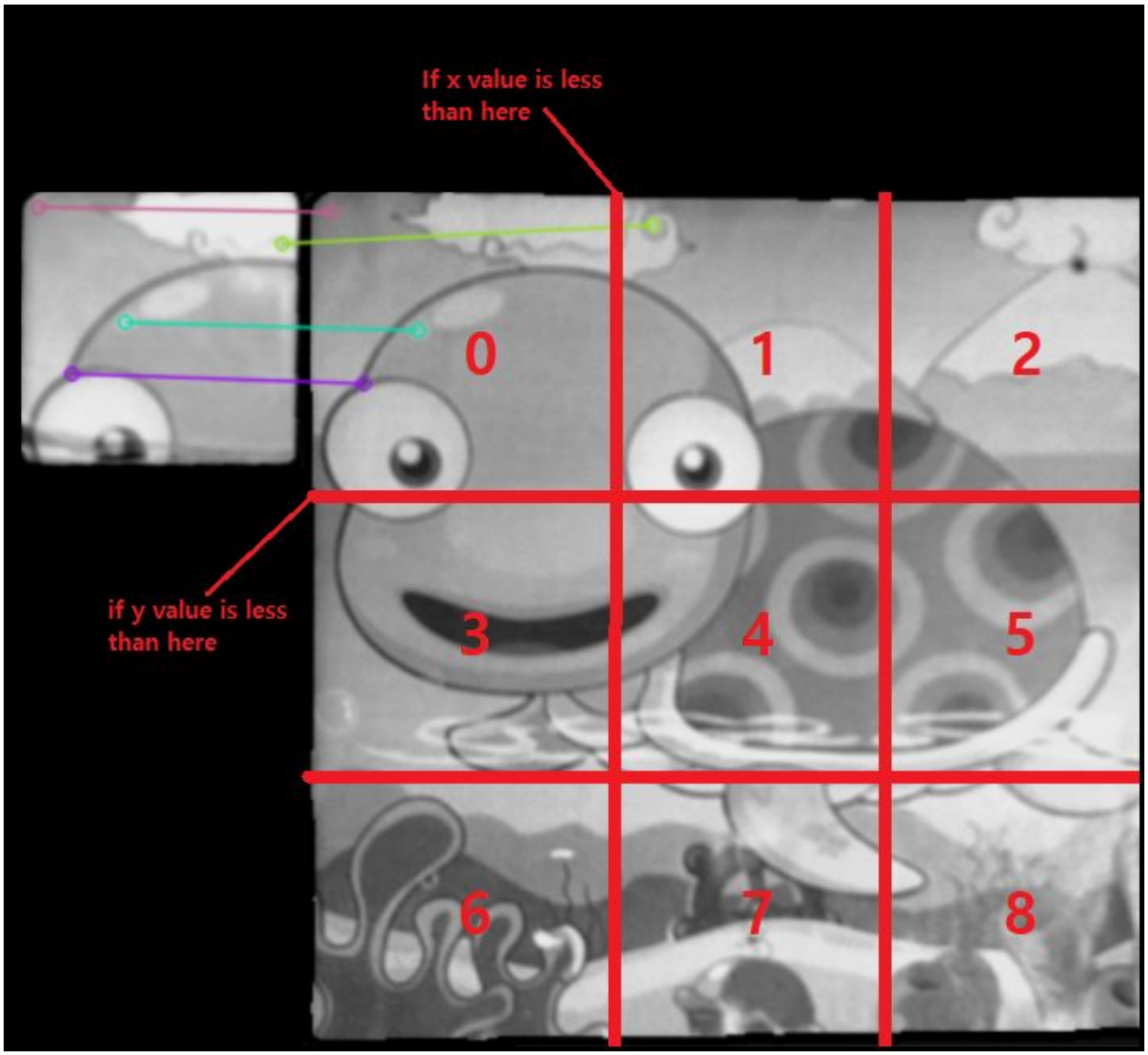


Figure 24: A piece being classified into “piece 0”

As we can see, three of the matched points fall within the range of “piece 0”, with one matched point falling in “piece 1”. The piece will then be voted to identify as piece 0.

This process will be repeated for every puzzle piece (as previously identified via contour area).

After this is completed, an array of size 3x9 array can be created:

```
[ {piece0centerx, piece0centery, destination}
  {piece1centerx, piece1centery, destination}
  {piece2centerx, piece2centery, destination}
  {piece3centerx, piece3centery, destination}
  ...]
```

At this moment, the array is sorted by the order the contours were found in. It must be rearranged such that it is in the correct placement order (ie, top left piece, top middle piece, top right piece, etc.)

This is done through the line:

```
sortedanswers = answers[answers[:, -1].argsort()]
```

After arranging the rows by the “destination” values, the array will be returned to the serial robot for in order for the pixel coordinates to be converted into the robotic coordinates.

A display of all the puzzle pieces and their locations will also be outputted for inspection by human operators.

Figure 25: Completed analysis output for human operator inspection

4.3.3 Everyone (Conveyor Belt):

The conveyor belt will be one of the final components that will be worked on since it will require a DC motor driver and requires a separate power supply of 24VDC to run. This DC motor driver will be connected to a slave Arduino since the all of the header I/O pins on the Raspberry Pi 3 are taken by the SlushEngine board. The other half of the L298A Dual H-bridge was used to control the conveyor belt but with a 12V 1A power supply, which in turn caused the conveyor belt to run slower than expected. The placement of the IR sensors were predetermined as the points where the serial robot could place the piece and where the delta arm could reach high enough and far enough to position above the piece. The placement took some time as these IR sensor placement would be fixed once they are placed. The Dual H-bridge was shared between the vacuum pump and the conveyor belt, so in the Arduino programming it was made to run one motor at a time and never run both as the power of the two motors would be split and affect the suction of the vacuum pump and the speed of the conveyor belt, making them ineffective.

4.3.4 Denisan Chandrakumar (Delta Robot):

Getting the Delta robot to communicate with the Raspberry Pi required packages to be installed in the proper sequence. The Raspberry Pi needed to enable SPI and I2C communication buses to properly use all I/O ports. The Raspberry Pi also needed the SlushEngine software from GitHub, to communicate to the SlushEngine board. To interface with the SlushEngine board and send commands, Python 3 IDLE was installed as another package. Using the Python 3 IDLE program, we were able to import the downloaded SlushEngine package and establish the defined board, motors, and other I/O connect to the SlushEngine board.

Currently the control of individual motors have been defined, there is controlled movement in the Z-axis due to the synchronous movement in all three motors, for the same distance at the same time. The inverse kinematics code that was provided for the delta robot does not have the right parameters for the delta robot being used. As a result, the code does not function properly and is unusable in the current state. The effort has made to reach out to Dr.Gao and figure out the correct parameters for the Delta robot. Additional code provided by Dr.Gao was useful in understanding how the Delta robot works and provided functions to setup and calibrate the robot.In addition, Dr.Gao helped us with a better understanding of how the inverse kinematics worked and how to obtain the four specific parameters(e, f, re, and rf) for the delta robot. Also, the producers of the SlushEngine board have been contacted to gather more information on the discontinued model and get a datasheet with all of the specifications of the board. Although by the end of the term no response was given, the conveyor belt was instead moved to the H-bridge. The suction mechanism for the pickup of the puzzle pieces is already in place using 12V DC vacuum pump connected to a 12VDC 1A power supply. When energized, the vacuum pump will activate and create suction. When the power de-energized, the vacuum pump will stop and slowly release the suction. A progress involving using a relay is being worked on to reverse the current and create a force to blow. The suction mechanism will not be controlled

using the Raspberry Pi due to limited I/O header pins leftover after connecting to the SlushEngine board and will be done using an Arduino microcontroller.

4.4 Testing/Manufacturing of Project:

4.4.1 Serial Robot:

The Dobot did not come with limit switches so pinpointing the home position and getting the robot to go there every time was becoming problematic, so markings were placed on the robot to indicate a proper home position that suited the need of our project. The robot would need to be shut off every time in order to set the correct initial conditions so that there could be more testing of the functionality of the robotic arm. The next step was to test the accelerometers on the Dobot to see if the limit switches could be bypassed when setting a proper coordinate for the home position. The code for the accelerometer calibration can be located in *Figure 26*. below and even implementing this solution to find a proper home position for the Dobot proved to be an issue as well. The coordinates could be properly calibrated and was able to print out the proper results. The issue came into play when the calibrated coordinates were sent to the robots move function. The robot began to do move and crunch the motors in a way that was not normal. Through more and more testing it was found out that the correct home position to which the Dobot would actually view as home was (160,0,215), so the arm was placed in the position that the markings were set for and that was considered (160,0,215). There is still no where in the code that indicates that this should be the proper home position, so all of our inverse kinematics calculations are based of this coordinate. During testing of the inverse kinematics it was noticed that sometimes that the x,y, and z coordinates programmed into the robot would not move to the proper coordinates and would deviate quite a bit from the position that it was supposed to be at. The robot was able to move to the point (150, 0, 90), so that is 15 cm in x, 0 cm in y, and 9 cm in z which from the home position of (160, 0, 215) is placed nicely on top of the puzzle platform. The trouble came in when the coordinate was moved 1 cm to (140,0,90), in theory the robot should move in x 1 cm, but in reality the robot would go somewhere completely different. This is still an issue and it cannot be determined whether it is a hardware issue due to the motors or limit switches or if there is an issue with the inverse kinematics code. There needs to be more testing on this.

```python
def InitializeAccelerometers(self):
    print("-------------")
    print("Initializing accelerometers")
    ret = (0, 0, 0, 0, 0, 0, 0)
    while not ret[0]:
        ret = driver.GetAccelerometers()
    accelRearX = ret[1]
    accelFrontX = ret[4]
    rearAngle = piHalf - driver.accelToRadians(accelRearX, accelOffsets[0])
    frontAngle = driver.accelToRadians(accelFrontX, accelOffsets[1])

    self._baseSteps = long(0)
    self._rearSteps = long((rearAngle / piTwo) * rearArmActualStepsPerRevolution + 0.5)
    self._frontSteps = long((frontAngle / piTwo) * frontArmActualStepsPerRevolution + 0.5)
    driver.SetCounters(self._baseSteps, self._rearSteps, self._frontSteps)
    print("Initializing with steps:", self._baseSteps, self._rearSteps, self._frontSteps)
    print("Reading back what was set:", driver.GetCounters())
    currBaseAngle = piTwo * self._baseSteps / baseActualStepsPerRevolution
    currRearAngle = piHalf - piTwo * self._rearSteps / rearArmActualStepsPerRevolution
    currFrontAngle = piTwo * self._frontSteps / frontArmActualStepsPerRevolution
     startXYZ = DobotIK.get_cartesian_coordinate_from_angles_using_forward_kinematics(currBaseAngle, currRearAngle, currFrontAngle)
    startXYZ = DobotIK.get_cartesian_coordinate_from_angles_using_forward_kinematics(0, 90, 0)
    print('Current estimated coordinates:', startXYZ)
    self.update_position_based_on_cartesian_coordinate(int(startXYZ[0]),int(startXYZ[1]),int(startXYZ[2]))
    print("-------------")
```

Figure 26: Code for initializing the accelerometer calibration

Moving to the coordinate system conversion from the pixel coordinate system to the millimeter system there was some issues with the conversion. The conversion was almost perfect, but the issue is how accurate the pixel points are picked up when the picture is taken then sent over to the pixel to millimeter conversion function and how well that handles the conversion. Once the conversion takes place and the points are sent to the motors the robot does go to a general location in the vicinity of the puzzle piece but it is not directly over the pice like it should, the difficulty is where does the problem lie? Does it lie with the conversion or the inverse kinematics or the physical constraints of the robot or is there some intermittent problem with the control board. There still needs more testing in deducing the exact problem, but if it is the conversion function then a coefficient may need to be added so that the robotic arm deviation be corrected. The conversion function can be seen in *Figure 27* below.

```python
def ConvertPixelCartesian(px, py):
    cx = (ox-px)*mm_pix
    cy = (py-oy)*mm_pix
    return (cy, cx)
```

Figure 27: Coordinate Conversion

There were several points in which the robotic arm was physically constrained, the arm was considered physically constrained when a crunching noise was made by the motors. When this happened the motors became displaced and therefore the entire position of the robot was displaced. Also certain coordinate points cause the robot to do strange movements as it traversed the plain. Any point greater than 150 in x would cause the robot to move somewhere it should not. This caused the workspace to be considerably smaller than when first anticipated.

4.4.2 Image Processing:

After installing these packages, pattern recognition code was developed based on sample code and other research done online.

Through research, several methods of pattern recognition was attempted: Brute-Force Matching with ORB Descriptors, Brute-Force Matching with SIFT Descriptors and Ratio Test, FLANN based Matcher.

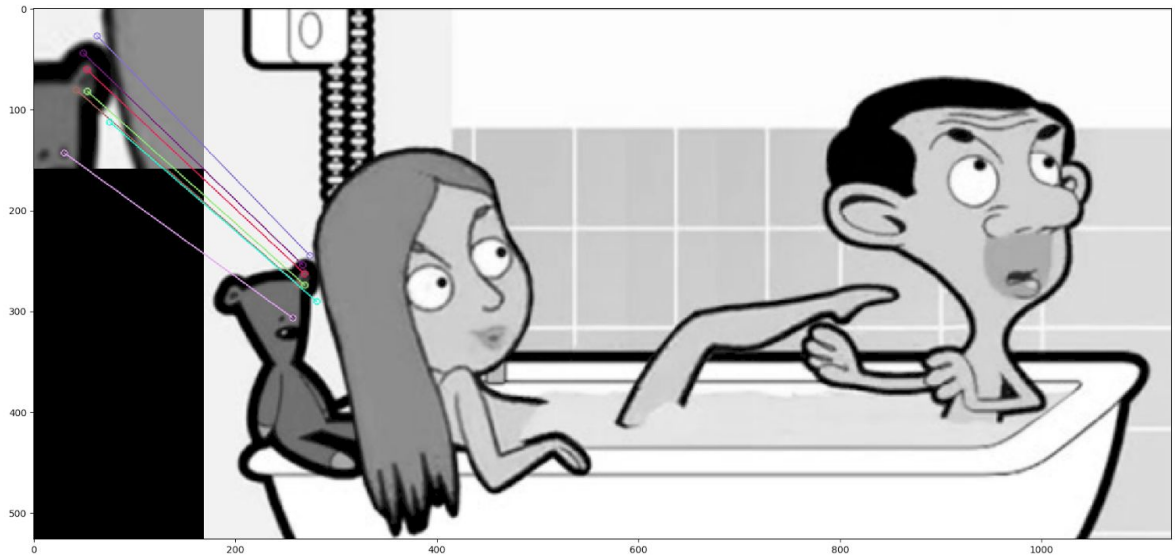


Figure 28: Pattern Recognition using SIFT on a Trial Image 1

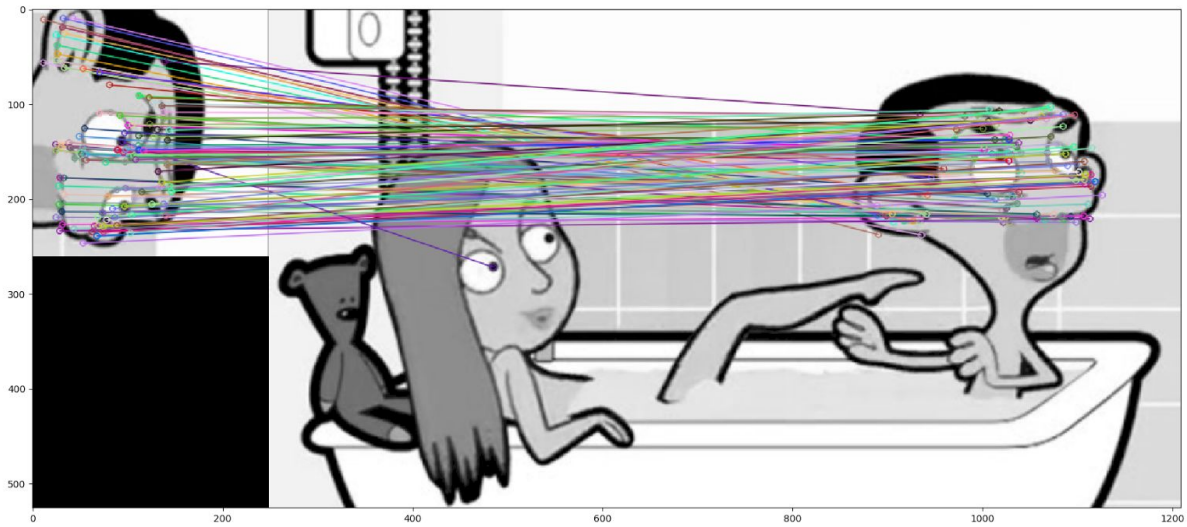


Figure 29: Pattern Recognition using ORB Descriptors on Trial Image 2

When installation of the additional package was completed, a test of the pattern recognition software was done on a cartoon scene before testing on pictures taken by the camera.

As can be seen, the pattern detection algorithm works perfectly when comparing two images with identical pixels. However, the results are slightly less accurate when dealing with images that are not identical in pixels. This deviation would be caused by inevitable lighting and angle changes when taking pictures with a camera.

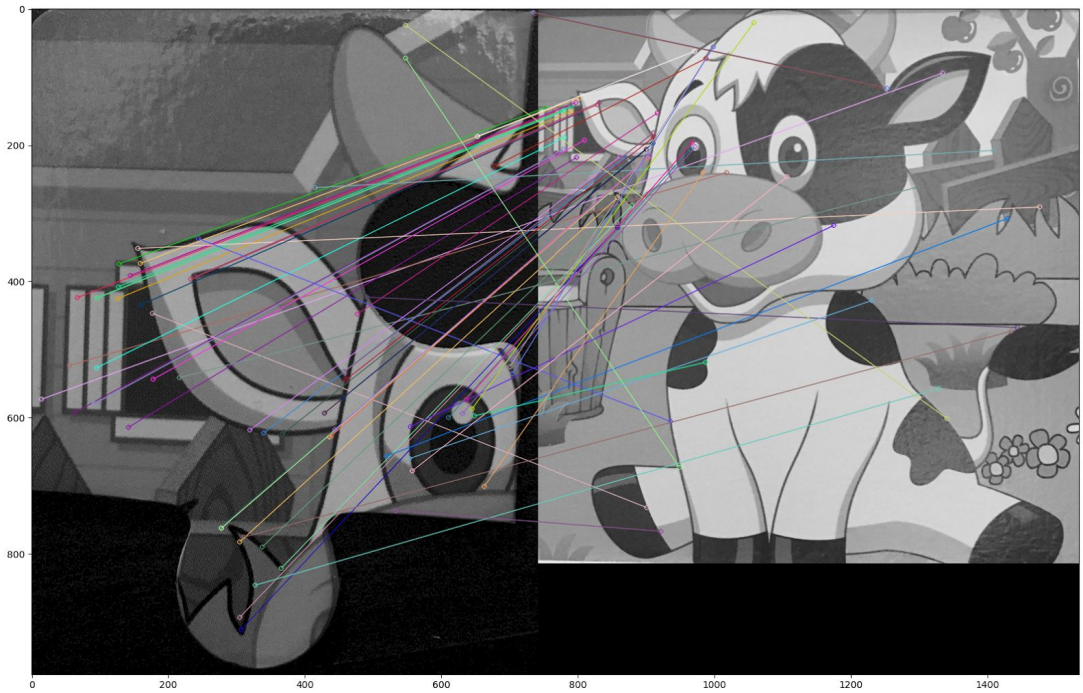


Figure 30: Pattern Recognition on a Actual Puzzle

As can be seen, when the target of image recognition are pictures taken from a camera, the results are more inaccurate and many false positives are being made.

4.4.3 Conveyor Belt:

The conveyor belt had a number of issues at the start with the main concern being that the plastic band used to connect the motor to the other end of the roller was not the right size since the plastic was not stretching and picking up the end of the roller. As a result, the conveyor belt speed was inconsistant and varied in speed often. This was an issue as the puzzle piece on the conveyor belt took far too long to travel from one end to the other end. In addition to the slow speed, sometimes the conveyor belt would jerk a bit past the sensor affecting the ending position of the puzzle piece in turn affecting the suction cup placement on the puzzle piece. To fix the plastic band, it was decided that high friction tape(electrical tape) would be used to add friction between the roller and plastic band. The roller and the plastic band was covered in the electrical tape and tested to insure that there was no loss of traction in the conveyor belt. The

conveyor belt was able to run a constant speed without slip. The improved conveyor belt is shown below in *Figure 31*.

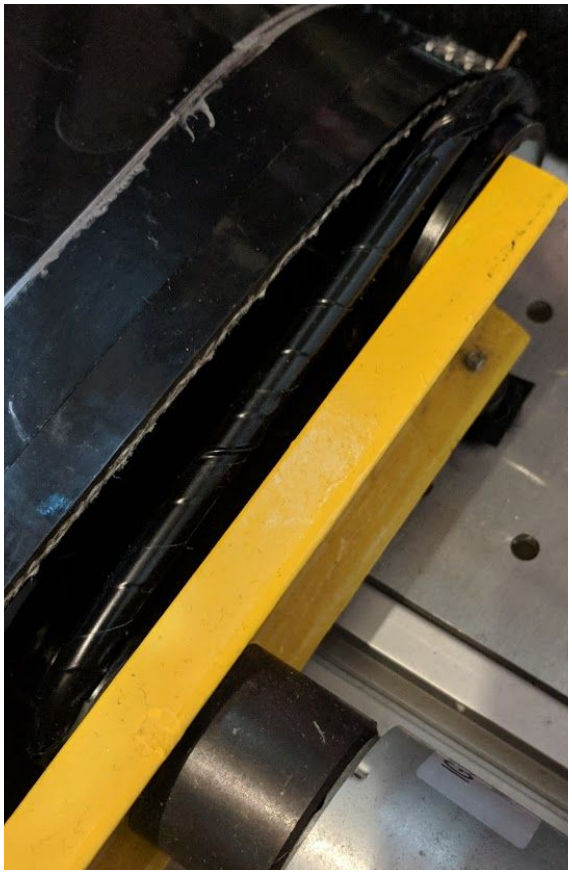


Figure 31: Conveyor belt after fixes

The IR sensors on the sides of the conveyor belt had a limited maximum range that would barely detect the puzzle since it would be a bit out of range. Although the sensors could be calibrated using a Phillips screwdriver, the max range was not within the requirements. The IR sensors needed to be moved closer to the edge of the conveyor belt to within range of detection and consistent throughout the runs. Repositioning of one of the IR sensors is shown below in *Figure 32*.

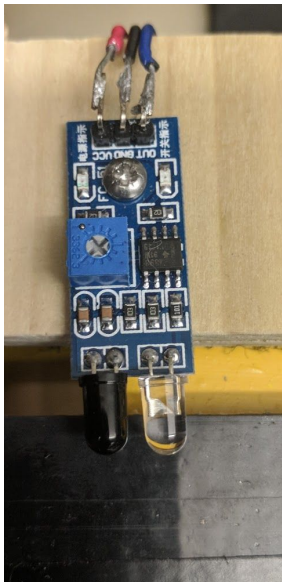


Figure 32: Repositioned IR sensor

Also, to ensure that the puzzle piece is always centered, the conveyor belt was disassembled to add guidelines on both sides using a silver Sharpie marker. The silver guidelines were measured on the conveyor belt and marked in a straight line parallel to the length of the conveyor belt. The silver guidelines added to the conveyor belt can be seen below in *Figure 32*.

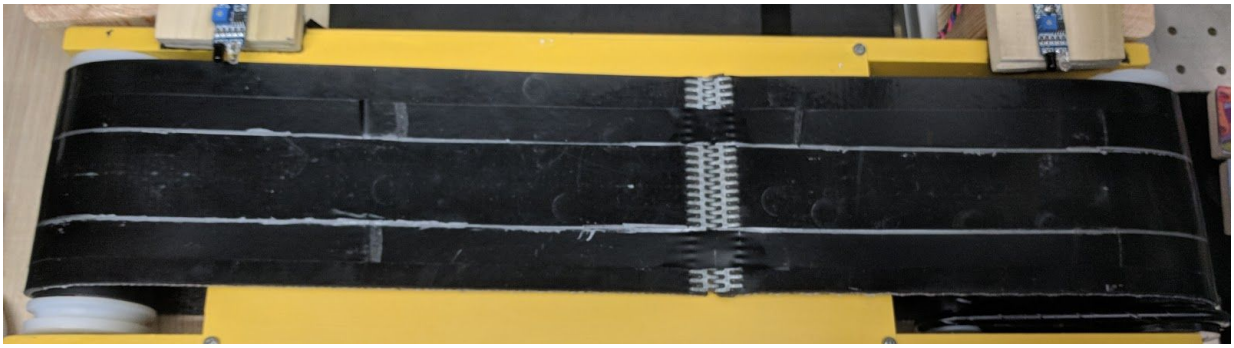


Figure 33: Conveyor belt with silver guidelines

4.4.4 Delta Robot:

The delta robot had no feedback since, the delta robot arm got in the way of the Pi camera that was to help place the puzzle pieces on the surface of the delta robot. As a result, it was decided that the placement of the puzzle pieces would need to be hardcoded, since we could not find another feedback system for placing the puzzle pieces in time. The change made the precision of the delta robot essential since the puzzle pieces needs to be placed in the same positions everytime the code was ran. Sample code is shown below in *Figure 34*.

```
for x in range(1, 10):
    while(read != 1):
        read = int(arduinoSerialData.readline())
        if x < 7:
            goConveyor()
        elif x >= 7:
            goConveyor2()
    read=0
    arduinoSerialData.write(bytes(read))
    if x == 1:
        print (moveDelta(-54, -44, 95))
        time.sleep(1)
        print (moveDelta(-54, -44, 115))
        time.sleep(3)
        print (moveDelta(-54, -44, 95))
        time.sleep(1)
    elif x == 2:
```

Figure 34: Snapshot of the code ran on delta robot

Another issue that was run into was during the calibration process of the delta robot. One of the limit switches that were used for the calibration of the delta robot were broken. Dr.Gao was contacted to address this issue and since the limit switches are very small, they would be difficult to replace. It was decided that an alternate method would be used to calibrate the robot. The alternate method consisted of identifying the center of the platform and wiring one of the limit switches in parallel with the broken switch. Using a center point, all three arms would travel the same distance in the calibration sequence and this would result all three limit switches to be triggered. Once the method was devised, it was quickly put to the test and worked exactly as expected. The center point was marked on the velcro with a silver sharpie marker and was always positioned there before run any code. A closer examination of the broken limit switch is shown below in *Figure 35*.

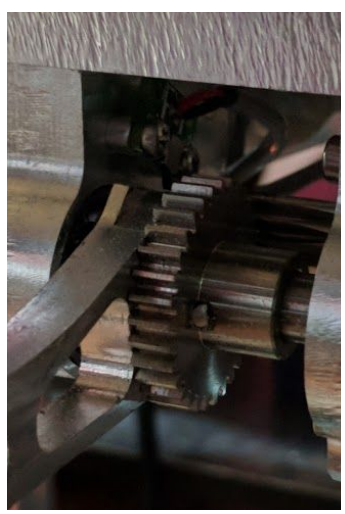


Figure 35: Closer look at the broken limit switch

After getting the positions of all nine puzzle pieces and testing with the puzzle pieces, came the task of the going to the pickup location on the conveyor belt. The issue was that the inverse kinematics had certain boundaries and once those boundaries are crossed, the inverse kinematics code would give back values of [0.0, 0.0, 0,0]. The code was scanned through multiple times to find where the boundaries were set and seeing if it can be modified to exceed the boundaries. The code couldn't be modified to the exceed the boundaries. Since, modifying the inverse kinematics didn't work, it was decided that the pickup sequence will be done manually. The manual movement of each motor to move to an area above the pickup location involved the arm being high enough to cross the edge of the conveyor belt without knocking the puzzle piece on the conveyor belt. After getting the arm across the conveyor belt with enough height, then come the issue of lowering the arm to the puzzle piece without moving the puzzle pieces location on the conveyor belt. This proves to be very difficult since moving all of the motors down, does not go directly down instead it goes downward in diagonal path since the arm is already extended in non-center position. If the arm was centered, moving all three motors down will result in the platform moving directly down.

4.5 Future Work:

4.5.1 Serial Robot:

The Dobot was a Kickstarter project initially so yes it was a fully functional product yet there may still be some issues with its initial product, like it was discovered that are no limit switches on the Dobot so calibration of the motor position was really difficult. So either replacing this system with a newer model might be an option if there is indeed a hardware problem. That being said if there is a software problem then that can be further diagnosed and solved. Possible software issues may include the inverse kinematics code, coordinate conversion code, as well as any of the motor driver codes that were found on Github[13]. In order to fix the inverse kinematics code some more diagnostics need to be done first in order to figure out exactly what the problem would be, there may be a possibility that the difference function may be the source of issue or even some of the other functions that are being called from other files. The issue with the coordinate conversion code is that there is a deviation from converted pixel value and where the robot is actually going, this could also be a hardware issue but assuming it is not a way to improve this would be add a coefficient to the conversion function so that it will not deviate from its desired location being sent from the camera. Finally the issues with the drivers could be many things as this has not been looked at because of shortage of diagnostic time, but nonetheless this still could be a potential issue, but it is unlikely. The stepper motor attached to the end effector of the robot was not very efficient for rotation because everytime the code was applied to rotate the motor for rotation it would rotate 90 degrees then automatically rotate back to its original position and not lock like it was supposed to, this could be a potential issue with the motor driver that was attached. All in all the robot functions as it should with a few minor hindrances.

4.5.2 Image Processing:

At the current moment, three things still need to be improved on for the vision recognition system.

1. Remove false positives
2. Verify that the votes are correct.
3. Obtain rotation angle

Currently, code that remove false positives already exist, however the effects cannot be shown in the display window. This is because only the coordinates are removed from the extracted array and not the found keypoints themselves. Another issue is the fact that the false positive points being eliminated are based off of the majority of “votes” (see section 4.3.2). If the majority of votes are incorrect to begin with, the entire analysis will be wrong regardless and the “correct” points will be eliminated instead.

Another issue that exists in the current code is that there is no way to verify the validity of the conclusion that it comes up with, this could perhaps be solved with analyzing all the puzzle pieces again with another pattern recognition algorithm like ORB descriptors or SURF. How discrepancies should be handled is another issue that must be put into consideration.

As of now, the code is unable to identify the rotation angles needed for the robot to correct the orientation of the puzzle piece. Again, some code for solving this already exist, but its usefulness is limited by its ability to remove false positives. This is because a rotation matrix is obtained via the sets of identified matching keypoints, if even a single false match exists, it would skew the code’s ability to obtain the correct rotation matrix. There is also an issue with slightly different values being obtained when using the rotation matrix.

4.5.3 Conveyor Belt:

The conveyor belt has been improved since the beginning of the term but, it can be improved further or replaced with a better conveyor belt. The fixes made a short-term repairs since the tape will eventually start peeling off the roller and the plastic band. A recommendation was made by Tom to purchase a new conveyor belt for $400 which was over the budget that was allowed for the group. The new conveyor belt was made by the same founders of Dobot, the serial robot we use in our project. The Dobot conveyor belt has controls for adjustable speed and a distance sensor. This conveyor belt would have made the Puzltron System better but, due to the lack of time to implement it was decided to stick with the conveyor belt provided.

4.5.4 Delta Robot:

The delta robot stills needs work to make it more consistent. The limit switch on one of the delta robot arm needs to be replaced. They may require a lot of time as the delta arm would need to be deconstructed and a new limit switch will need to be purchased of the same model to ensure

that the limit switch do not differ in consistency. The inverse kinematics code will need to be modified to determine why the inverse kinematics will not exceed a certain value. Also, the industrial ports and the digital I/O on the Slush Engine board are still undetermined and no response from the group that created the SlushEngine board, since the board given us is the first edition board and there is no datasheet reference for that specific board.

5. Discussion:

5.1 Viability of the Project:

This project is heavily weighted towards the software and visual recognition of the system, since both robots are currently built and operational. This project is viable, but is completely contingent on the system recognizing the individual puzzle pieces. That is why, distinctive puzzle pieces will be used at first. Once the software is capable of working with these pieces, regular jigsaw pieces will be implemented.

5.2 Challenges Faced and Overcame:

Problem:	When the Dobot packaging was opened, it was discovered that the robot had already been constructed.
Solution:	It was confirm with datasheets and other sources to ensure that the robot was constructed properly and not missing any parts. The robot as tested using the on-board controller to ensure that the robot's mechanical components are intact.

Problem:	It was decided that the Dobot and Delta robot would be placed in the basement of ETB B111 and every Thursday we would come to the class and work on the capstone. After two weeks of working on the capstone, the Raspberry Pi that was in our box, suddenly went missing. Confused to where the Raspberry Pi went as our code and setup for the Pi were wasted. Later our Raspberry Pi was returned.
Solution:	Decided to move the robots to a better, more secure location. Instead of meeting up on Thursday to work on the capstone in the TR room, it was decided that it would best to work on the capstone on Saturdays at the new location.

Problem:	We had bought a DP to HDMI cable that would connect the powered Raspberry Pi to the school monitor. After we received the cable, we connected the two and saw nothing on the monitor. We realized after multiple attempts to get the display working, that the cable was broken.
Solution:	Denisan decided to bring his monitor to the lab to use HDMI to HDMI connection and display the Pi. It was a success as the monitor auto-detected the resolution and displayed the setup. For future consideration, the resolution output of the Raspberry Pi was lowered from 1080p to 720p which allows the Raspberry Pi to display on different monitors.

Problem:	The communication with the SlushEngine board relied on many external packages to be installed on the Raspberry Pi and needed the packages to be installed in a certain order.
Solution:	Figure out the proper order to install the packages. Doing the proper installation required the previous packages to be uninstalled and reinstalled in the proper order without any fail.

Problem:	The power supply that was bought for the delta robot was rated at 12VDC 12.5A, the SlushEngine board can only handle up to 35VDC 5A.
Solution:	Decided to improvise with a laptop adapter that was no longer needed. This laptop adapter provided 19VDC 5A and meet the requirements perfectly for the SlushEngine board. After removing the end of the connector and exposing the leads and are able to identify the positive and negative terminal through online sources, connect these leads to the board.

Problem:	OpenPLC works for a bit on the Raspberry Pi and the spontaneously stops working.
Solution:	After struggling with this issue for over 5 hours, it was decided that since this software was not working properly with the Raspberry Pi that another software would be used instead.

Problem:	PyCharm consuming all of the Raspberry Pi's 1GB of RAM
Solution:	PyCharm was moved to another computer with more processing power and the image recognition will be coded on that computer and once that is completed it will be moved over to the Raspberry Pi where it will be integrated into the hardware.

Problem:	An unidentified external monitor, keyboard, and mouse connected to the Delta Robot and suggests potential tampering between work sessions
Solution:	Complete removal of unidentified external monitor, keyboard, and mouse and the Delta robot was taken out of the lab and to a team members house for safe keeping and development.

5.3 Lessons Learned:

5.3.1 Serial Robot:

The dobot was a very functional robotic tool to work with for this project, the only issue that was had with it was that it was a first generation Kickstarter model and that it was pre built so there could have been some problems because of those facts. If this project were to be done again a newer generation model should replace the current one and the students that are doing the project should build it so that they know exactly the condition of the robot as they are the ones that built it. Aside from that working with the Dobot was both fun and educational as there were many different ways it could have been coded. It could have been coded straight from the Arduino Mega that came with it. Raspberry Pi 3 was integrated with the Dobot, so then a blank program had to be flashed on the Arduino Mega so that it could be coded through Python on the Raspberry Pi 3. Learning how to code in Python for this project was also interesting as it shares the same syntax as C++ but there are some nuances that are different and getting used to that was both educational and frustrating.

5.3.2 Image Processing:

One thing that was very crucial to the development of the image processing software that was very much overlooked prior to development was maintaining consistency in the lighting of the work space. Any lighting changes from picture to picture, test environment to test environment would severely affect the sensitivity values for both contours and the SIFT algorithm.

An LED light strip was chosen in the end to control the lighting of the workspace such that a consistent quality can be achieved.

However with this addition, there now exist a problem where the puzzle pieces themselves are now far too reflective, to a point where the illustrations on the piece are washed out. In order to solve this problem, the puzzle pieces were scanned and printed out on printer paper, chosen for its non reflective surface. The puzzle pieces then had their surface covered with the printed out illustration in their identical orientations.

5.3.3 Conveyor Belt:

Although the conveyor belt was simple to hook up to the H-bridge and control, the conveyor belt did cause many issues with its performance. The conveyor belt was not an effective component and often lacked in speed. If this project where to be repeated again I would recommend that the conveyor belt is replaced with a better conveyor belt. The surface of this conveyor belt was very slippery and often caused puzzle pieces to slide when puzzle pieces were either dropped off or picked up. A conveyor belt with a constant speed and a non-slip surface would be much better for the Puzltron system.

5.3.4 Delta Robot:

The delta robot was very impressive with its speed and precision. The SlushEngine board had the initialization of the stepper motors in the SlushEngine library that made the entire process of setting up the delta robot easier. The only issue that was faced working with the inverse kinematics code was understanding the parameters that would fit the delta robot given. Luckily, Dr.Gao knew exactly where to locate all these parameters and taught us how the inverse kinematics code works. Also, the workaround for the delta robot calibration is only temporary since the broken limit switch was wired in parallel with another limit switch. If that another limit switch were to break. Then the delta robot calibration step could not continue since it waits until the switch is triggered to begin running the rest of the code. That would be highly recommended that any group working with the delta robot consult with Dr.Gao.

5.4 Practical Applications:

5.4.1 Automated Assembly Line:

This project models an automated assembly line, through the use of visual recognition software. In an ideal factory setting, parts would be laid out in a workspace, and a camera will constantly monitor incoming parts and determine the order in which to assemble. The sorted parts will then be moved one by one onto a conveyor belt towards the next part of the production process. Depending on the complexity of the assembly, the appropriate type robot should be chosen. Visual recognition is fundamental part to the future of automation and that is why it is the primary focus of this project.

5.4.2 Learning Factory:

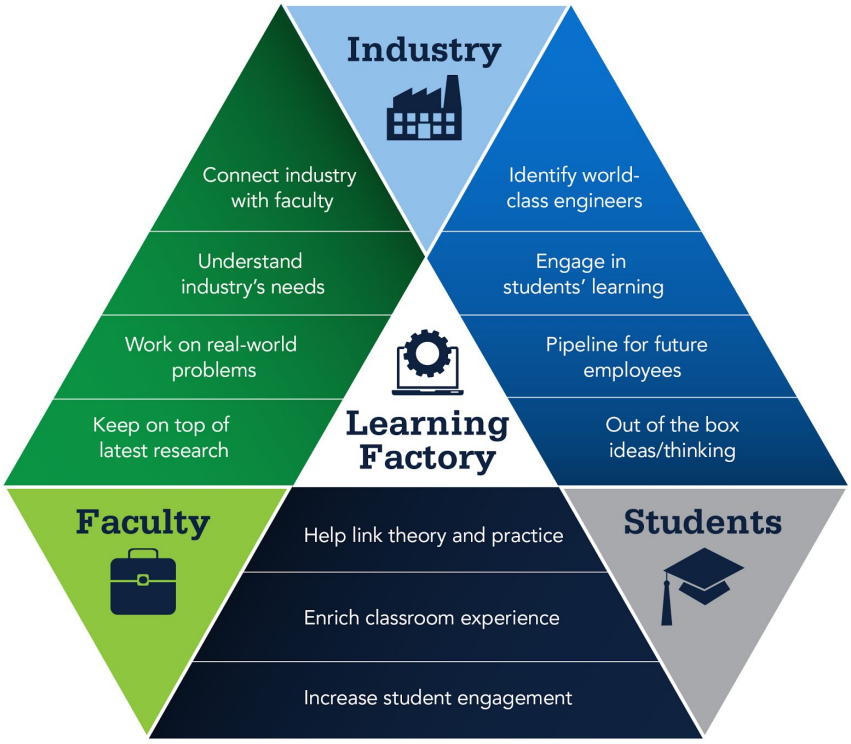


Figure 36: Learning Factory

Another application of this project is the "learning factory" and how it can be used to teach younger students that are interested in robotics and programming, on how they work together. Connecting industry, faculty and students together. Refer to *Figure 36* above [11]. This project can be applied in industry, to represent a model assembly, as mentioned before. High School students and younger students interested in engineering can build on this project and see how this system was integrated and make their own.

5.4.3 Silicon Valley Pizzeria:

In Silicon Valley, the integration of serial robot, delta robot and conveyor belt has been done and has been proven to work efficiently. They have created a "Robotic Pizzeria" that uses the delta robot to apply and spread tomato sauce on a pizza dough while workers sprinkle on the cheese and other toppings. The conveyor belt moves the uncooked pizzas down the production line towards the serial robot, which picks up the pizza and moves it into the oven where the pizza is cooked [12]. This is closely related to this project as it is essentially an identical system, except with a more practical application. While the Pizzeria system currently does not implement a visual recognition system, in the future it is possible to automate adding the cheese and toppings part of the process through the use of a camera.

5.5 Problem Solving Methods:

The most difficult part of the project will be the coordination between units in the system and the image processing required to identify puzzle pieces. General cooperation between each of the components also prove to be a complicated procedure. In order to effectively simplify these problems, it is important to tackle each issue at its most basic level, ensure each piece of the system works separately before integrating them together.

For example, it is important the coding for color identification or simple robotic movements are bug-free (error free, precise movement) before these codes are implemented to recognize patterns or pick and place puzzle pieces. This way, it will be easier to troubleshoot as the problem originating from software issues can be rooted out and instead, efforts can be focused on debugging the integration or hardware issues.

6. Conclusion:

The basic premise of the project aims to create a model assembly line using various components to complete a jigsaw puzzle. On a deeper level, this project also aims to help with educating other upcoming engineering students with theories of inverse kinematics, image recognition, communication protocols through this small scale assembly line. The fact that it is a physical object that can be manipulated and changed, promotes hands-on learning, as opposed to merely relying on theoretical knowledge via lectures and instructors. Though this project is currently only just ending its conceptual design phase, it will be moved on to coding and integration stages in 4TR3, with hopes of completion by December 2017.

7. References:

[1] N. Corporation, "Technical calculation service," in *Nidec Corporation*. [Online]. Available: http://www.nidec.com/en-NA/technology/calc/torque/conveyor_belt/. Accessed: Feb. 22, 2017.

[2]"Dobot_Specifications," in *Dobot*, 2016. [Online]. Available: http://www.dobot.cc/dobot-magician/specification.html. Accessed: Feb. 22, 2017.

[3] R. Williams, "DeltaKin," 2016. [Online]. Available: http://www.ohio.edu/people/williar4/html/pdf/DeltaKin.pdf. Accessed: Feb. 24, 2017.

[4] R. M. Murray, Z. Li, and S. S. Sastry, "A Mathematical Introduction to Robotic Manipulation,". [Online]. Available: http://www.cds.caltech.edu/~murray/books/MLS/pdf/mls94-complete.pdf. Accessed: Feb. 25, 2017.

[5]"Raspberry Pi 3 Model B - Raspberry Pi", *Raspberry Pi*, 2017. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Accessed: 26- Feb- 2017].

[6]"Delta Robot Kinematics - Tutorials". *Forums.trossenrobotics.com*. N.p., 2017. Web. 28 Feb. 2017.

[7] PROC TECH 4TR1 - Capstone Design Project 1 - Lecture 1: Intro, Slide 9. April 9th, 2017

[8]"Lynxmotion AL5D PLTW Robotic Arm Kit w/ FlowArm PLTW App", Robotshop.com, 2017. [Online]. Available: http://www.robotshop.com/ca/en/lynxmotion-al5d-pltw-robotic-arm-kit.html. [Accessed: 09- Apr- 2017].

[9]"pixy cmuCAM + Arduino", *Arduino Project Hub*, 2017. [Online]. Available: https://create.arduino.cc/projecthub/niftyjoeman/pixy-cmucam-arduino-3ac141. [Accessed: 09- Apr- 2017].

[10]M. Kaleemuddin S, "Performance comparison of Infrared and Ultrasonic sensors for obstacles of different materials in vehicle/ robot navigation applications", 2017. [Online]. Available: http://iopscience.iop.org/article/10.1088/1757-899X/149/1/012141/pdf. [Accessed: 09- Apr- 2017].

[11]"Penn State Engineering:The Learning Factory", Lf.psu.edu, 2017. [Online]. Available: http://www.lf.psu.edu/. [Accessed: 09- Apr- 2017].

[12]O. Zaleski, "Inside Silicon Valley's Robot Pizzeria", Bloomberg.com, 2017. [Online]. Available: https://www.bloomberg.com/news/articles/2016-06-24/inside-silicon-valley-s-robot-pizzeria. [Accessed: 09- Apr- 2017].

[13]GitHub. (2017). *maxosprojects/open-dobot*. [online] Available at: https://github.com/maxosprojects/open-dobot [Accessed 11 Dec. 2017].

[14]"OpenCV: Introduction to SIFT (Scale-Invariant Feature Transform)", *Docs.opencv.org*, 2017. [Online]. Available: https://docs.opencv.org/3.3.0/da/df5/tutorial_py_sift_intro.html. [Accessed: 11- Dec- 2017].

[15]"OpenCV: Contour Properties", *Docs.opencv.org*, 2017. [Online]. Available: https://docs.opencv.org/trunk/d1/d32/tutorial_py_contour_properties.html. [Accessed: 11- Dec- 2017].

8. Appendices:

8.1 Appendix A: Table 1: Bill of Materials:

List of materials required and their costs

Item Name	Quantity	Cost (per unit)	Total Cost	Source
Delta Robot	1	0	0	Btech

Dobot	1	0	0	Btech
Conveyor Belt System	1	0	0	Btech
Raspberry Pi 3	2	$54	$108	http://www.banggood.com/Raspberry-Pi-3-Model-B-ARM-Cortex-A53-CPU-1_2GHz-64-Bit-Quad-Core-1GB-RAM-10-Times-B-p-1041862.html?rmmds=search
Raspberry Camera	2	$38	$76	http://www.banggood.com/Raspberry-Pi-V2-Official-8-Megapixel-HD-Camera-Board-With-IMX219-PQ-CMOS-Image-Sensor-p-1057157.html?rmmds=search
IR Sensor	1	$6	$6	http://www.banggood.com/Mini-38KHz-IR-Infrared-Transmitter-Module-IR-Infrared-Receiver-Sensor-Module-For-Arduino-RPI-STM32-p-1066428.html?rmmds=search
Big red emergency stop button	2	0	0	Self supplied
Suction Cup	2	$6.65	$13.30	http://www.robotshop.com/ca/en/vacuum-gripper-cup.html

Pneumatic Pump	2	$20	$40	http://www.robotshop.com/ca/en/sfe-air-pump.html
Pneumatic Elbow Fitting	2	$3.27	$6.54	http://www.robotshop.com/ca/en/vacuum-gripper-elbow-fitting.html
Pneumatic Tubing	2m	$2.61	$2.61	http://www.robotshop.com/ca/en/2m-6-pneumatic-tube-blue.html
12 Volt Power Source	1	$44	$44	http://www.robotshop.com/ca/en/power-supply-12v-15a.html
Puzzle	2	$4	$8	Dollarama
Plexi Glass walls 12in x 24in	5 sheets	$15.25	$76.25	http://www.homedepot.com/p/LEXAN-12-in-x-24-in-x-093-in-Clear-Polycarbonate-Sheet-GE-33/202038063
Wiring (16 gauge, solid)	10m	-	-	Btech
ULine Trolley	1	?	?	Btech
Plywood Board	2m x 1m	~$30	~$30	Home Depot

8.2 Appendix B: Sample Code:

8.2.1 Serial Robot Sample Code:

```
#Piece 1:
for i in range(1):
  MoveToAndOrigin(180, 0, 102, 0, 0)

#Forward to Conveyor Belt
execute(freq2, increasing, freq2, 0, 1, 0, 0, 0)
```

```
execute(increasing, freq2, freq2, 1, 0, 0, 0, 0)
execute(increasing, freq2, freq2, 1, 0, 0, 0, 0)
time.sleep(2)
for i in range(1):
  MoveToAndOrigin(180,-25, 120, 1, 0)

#Backwards towards puzzle tray

execute(increasing, freq2, freq2, 0, 0, 0, 0, 0)
execute(increasing, freq2, freq2, 0, 0, 0, 0, 0)
execute(freq2, increasing, freq2, 0, 0, 0, 0, 0)

time.sleep(10)
```

8.2.2 Image Processing Sample Code:

8.2.2.1 Contour Finding Code:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2

centerx = []
centery = []


def findContours(img,min_threshold_area,max_threshold_area):
   filteredcontours = []
   blurred = cv2.GaussianBlur(img, (7, 7), 0)
   threshold = cv2.threshold(blurred, 40, 255, cv2.THRESH_BINARY)[1]

   kernel = np.ones((3, 3), np.uint8)
   closing = cv2.morphologyEx(threshold, cv2.MORPH_CLOSE, kernel, iterations=6)

   cv2.imshow('th',closing)
   cv2.waitKey(0)

   _,contours,_=cv2.findContours(closing,cv2.RETR_LIST,cv2.CHAIN_APPROX_NONE)
   #print "Number of countours:" %len(contours)


   for cnt in contours:
       area = cv2.contourArea(cnt)
       print area
       if (area > min_threshold_area) and (area < max_threshold_area):
           #print area
```

```
            filteredcontours.append(cnt)

    cv2.drawContours(img,filteredcontours,-1,(0,0,255),3)

    cv2.imwrite("contour.jpg", threshold)
     cv2.imshow('img',img)
     cv2.waitKey(0)
     return filteredcontours
```

8.2.2.2 Pattern Recognition Code:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2


def imageRec(img1,img2):

    # Initiate SIFT detector
    sift = cv2.xfeatures2d.SIFT_create()

    img1 = cv2.GaussianBlur(img1, (7, 7), 0)
    img2 = cv2.GaussianBlur(img2, (7, 7), 0)

    # find the keypoints and descriptors with SIFT
    kp1, des1 = sift.detectAndCompute(img1,None)
    kp2, des2 = sift.detectAndCompute(img2,None)

    # BFMatcher with default params
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des1,des2, k=2)
    # Apply ratio test
    good = []
    gooder = []
    for m,n in matches:
        if m.distance < 0.70*n.distance:
            good.append([m])
            gooder.append(m)

    pcs_pts = np.float32([kp1[m.queryIdx].pt for m in gooder])
    full_pts = np.float32([kp2[m.trainIdx].pt for m in gooder])

    print "pcs_pts"
    print pcs_pts
    print "full_pts"
    print full_pts

    height, width = img2.shape[:2]
    location =[]

    for m in full_pts:
```

```
        if m[0] < (width / 3): #0,3,6
            if m[1] < (height/3): #0
                location.append(0)
            elif (m[1] > (height/3))and (m[1] < ((height/3)*2)): #3
                location.append(3)
            elif m[1] > ((height/3)*2):#6
                location.append(6)

        elif (m[0] > (width/3)) and (m[0] < ((width/3)*2)):#1,4,7
            if m[1] < (height/3): #1
                location.append(1)
            elif (m[1] > (height/3))and (m[1] < ((height/3)*2)): #4
                location.append(4)
            elif m[1] > ((height/3)*2): #7
                location.append(7)

        elif m[0] > ((width/3)*2): #2,5,8
            if m[1] < (height / 3):  #2
                location.append(2)
            elif (m[1] > (height / 3)) and (m[1] < ((height / 3) * 2)):  #5
                location.append(5)
            elif m[1] > ((height / 3) * 2): #8
                location.append(8)


    print location
    totalvote = max(location, key=location.count)
    #totalvote = 1
    #print totalvote

    ''' #code that eliminates false matches, WIP
    for index,m in enumerate(location):
        print m
        if (m != totalvote):
            print("index: "+str(index))
            print("m: "+str(m))
            full_pts = np.delete(full_pts,index,0)
            pcs_pts = np.delete(pcs_pts,index,0)
    print location
    print full_pts
    print pcs_pts
    '''

    #cv2.drawMatchesKnn expects list of lists as matches.
    img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,good,None,flags=2)
    return img3, pcs_pts, full_pts, totalvote
```

8.2.3 Conveyor Belt and Pump Control Sample Code:

void loop() {

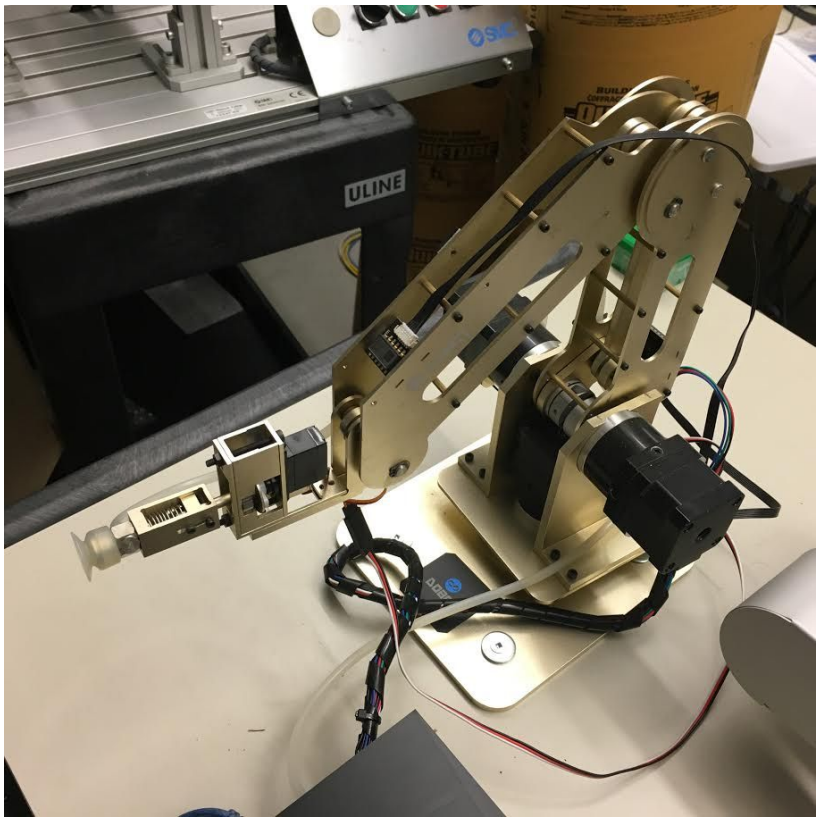
```
  // put your main code here, to run repeatedly:
     sensor1 = digitalRead(2);
     sensor2 = digitalRead(4);

  if(sensor1 == LOW && sensor2 == HIGH)
  {
    delay(5000);
    sensor1 = digitalRead(2);
    sensor2 = digitalRead(4);
     if((sensor1 == LOW && sensor2 == HIGH))
      {
       do{
          digitalWrite(10, HIGH);   //Run conveyor belt
          digitalWrite(11, LOW);   //Stop conveyor belt
          sensor1 = digitalRead(2);
          sensor2 = digitalRead(4);
       }while((sensor2 == HIGH));
       if((sensor1 == HIGH && sensor2 == LOW))
       {
         digitalWrite(10, LOW);
         digitalWrite(11, LOW);     //Stop conveyor belt
         Serial.println("1");     //Tell robot to pickup
         do
         {
           x = Serial.parseInt();
         }while(x != 2);
         while(x == 2)
         {
         digitalWrite(5, LOW);    //Run pump
         digitalWrite(6, HIGH);
         delay(2000);
         x = Serial.parseInt();
         }
         do
         {
           x = Serial.parseInt();
         }while(x != 0);
         digitalWrite(6, LOW);
       }
      }
   }
}
```

8.2.4 Delta Robot Sample Code:

```
calibrateDelta()
time.sleep(1)
for x in range(1, 10):
    while(read != 1):
        read = int(arduinoSerialData.readline())
        if x < 7:
            goConveyor()
        elif x >= 7:
            goConveyor2()
    read=0
    arduinoSerialData.write(bytes(read))
    if x == 1:
        print (moveDelta(-54, -44, 95))
        time.sleep(1)
        print (moveDelta(-54, -44, 115))
        time.sleep(3)
        print (moveDelta(-54, -44, 95))
        time.sleep(1)
    elif x == 2:
```

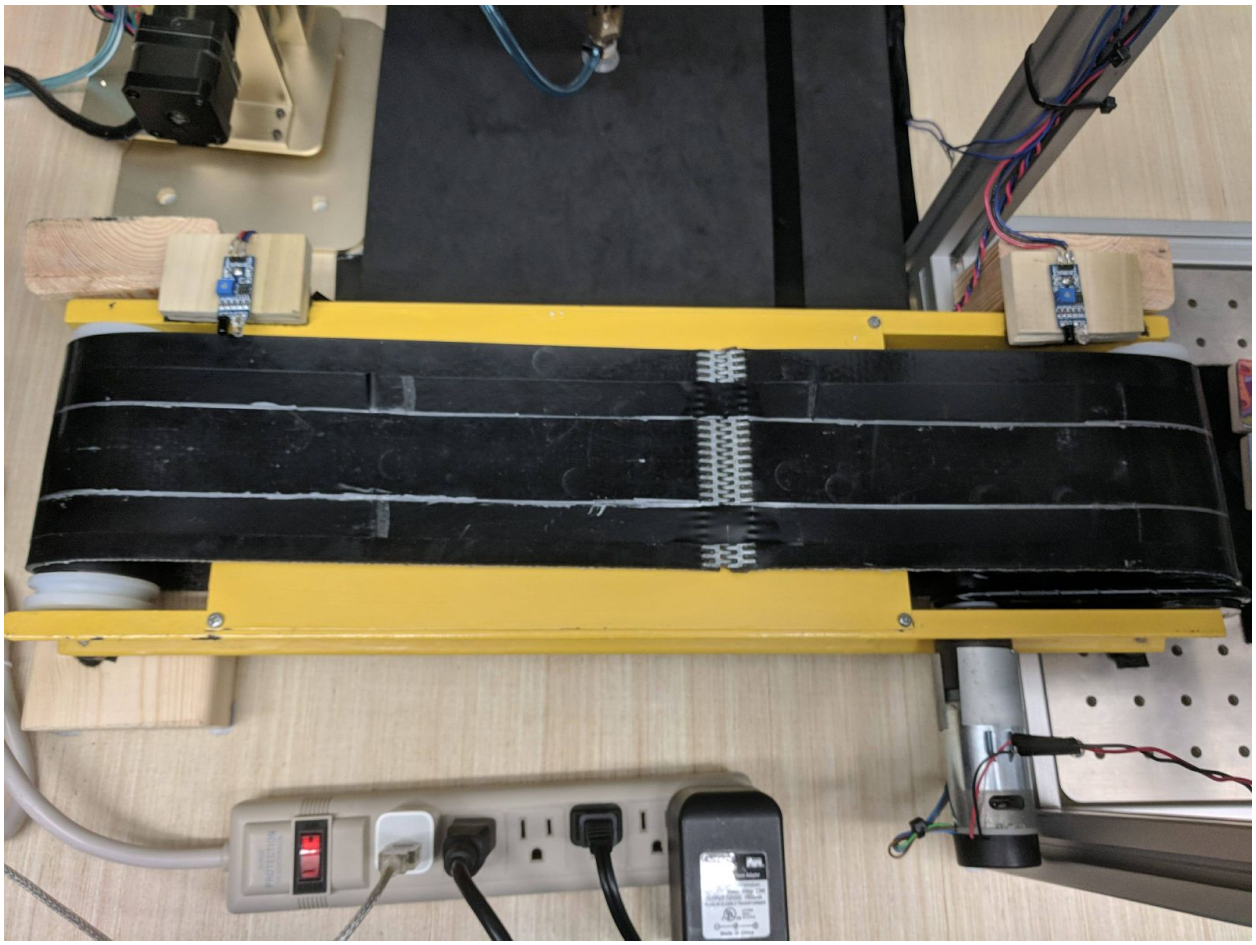
8.3 Appendix C: Pictures of Components:

8.3.1 Serial Robot:

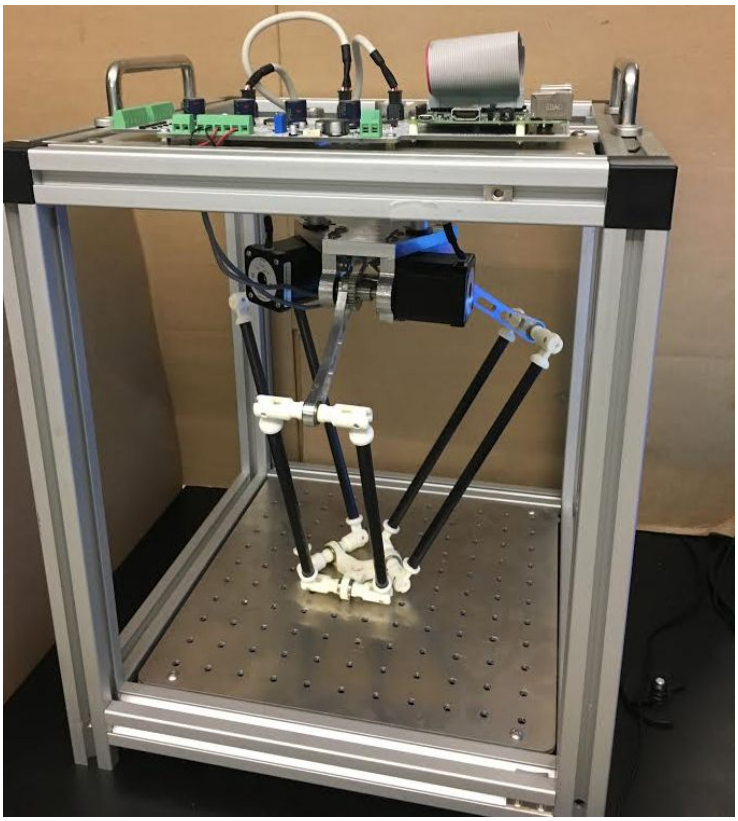


8.3.2 Image Recognition:

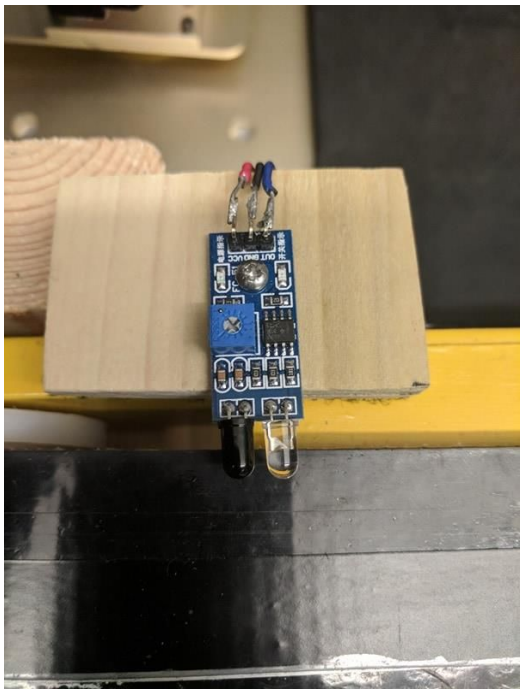
8.3.3 Conveyor Belt:



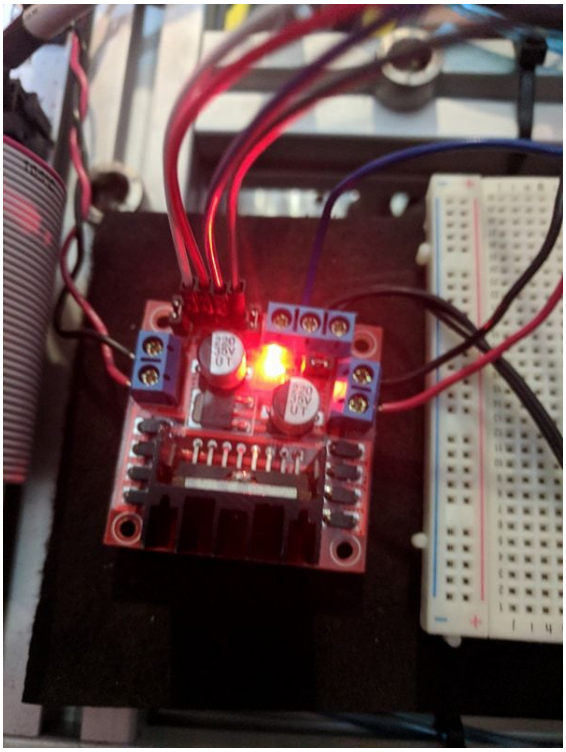
8.3.4 Delta Robot:



8.3.5 IR sensor:



8.3.6 Dual H-bridge:

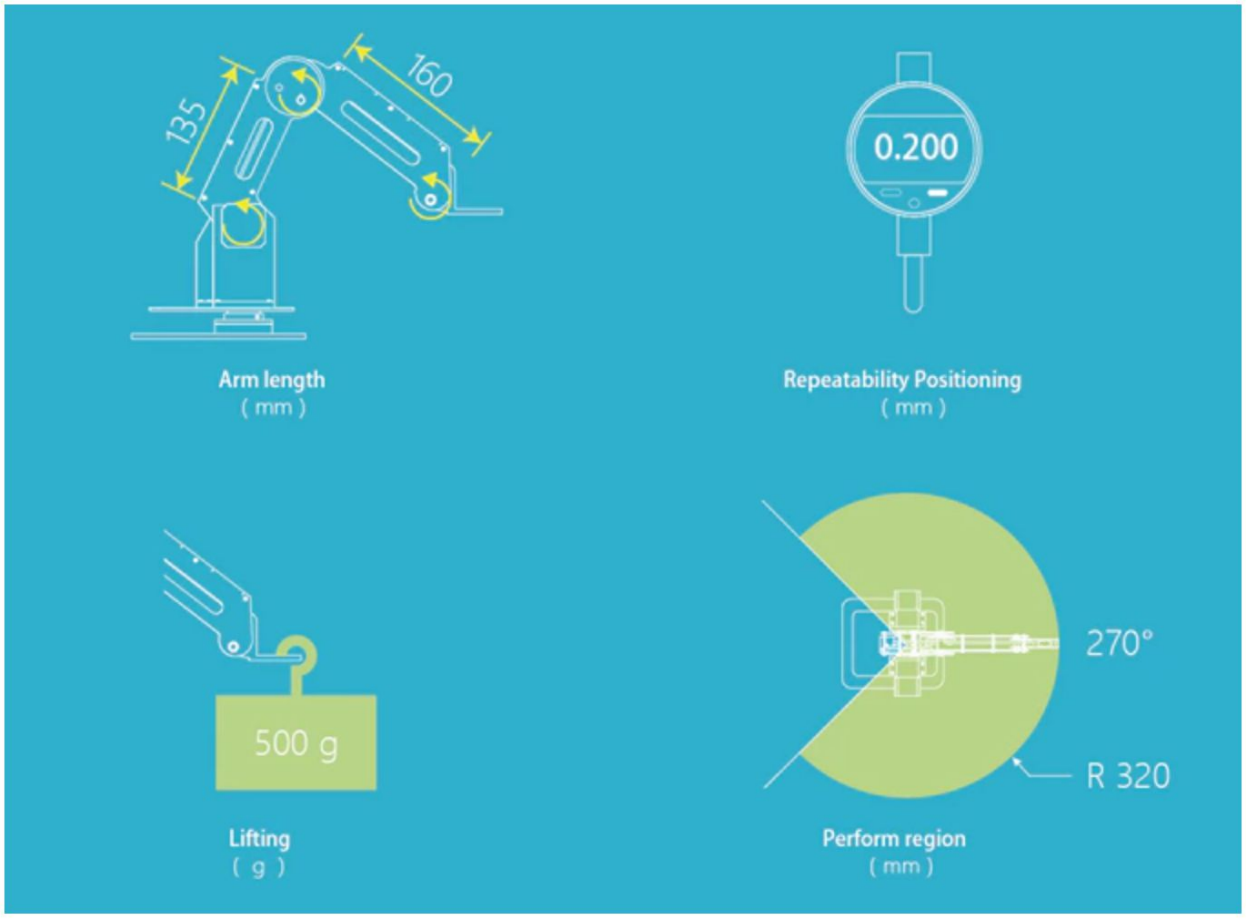


8.4 Appendix D: Table 2: Expense Report:

Date	Item	Price(tax incl.)
08/22/2017	Raspberry Pi V3 x2	$114.27
08/22/2017	Raspberry Pi Camera	$43.94
09/07/2017	IR sensor x2	$9.95
09/07/2017	E-stop button	$6.28
09/07/2017	60W Power Supply	$13.28
09/07/2017	Suction Cup	$6.65
09/07/2017	Pneumatic Pump	$20.00
09/07/2017	Pneumatic Elbow Fitting	$3.27
09/07/2017	Pneumatic Tubing	$2.61
09/21/2017	Puzzle x2	$2.26
09/21/2017	16gb MircoSD card x2	$38.40
10/21/2017	Pi Camera Case	$9.99
10/21/2017	Pi Case Enclosure x3	$31.08
11/23/2017	Velcro 15m	$35.67
11/23/2017	Plywood board	$24.39
11/23/2017	2” x 4” stud	$3.76
11/23/2017	Bolts and nuts	$6.23
11/23/2017	Hacksaw	$7.84
11/24/2017	Wiring (16 gauge, solid)	$8.29
11/24/2017	Laptop power supply	$32.45
11/24/2017	12V LED strip	$12.76
12/01/2017	**Total**	**$433.37**

8.5 Appendix E: Dobot Specifications:

Axis MoveMents	Working Range	Max Speed	Torque	Resolution	Drive
Axis 1	-135º-135º	200º/s	6.4N·M	0.01º	Stepper Motor
Axis 2	-15º-85º	200º/s	6.4N·M	0.01º	Stepper Motor
Axis 3	-20º-95º	200º/s	6.4N·M	0.01º	Stepper Motor
Axis 4	-90º-90º	0.06sec/60º	3.7kg·cm@7.4V	0.13º	servo



Number of Axes	4
Weight	3kg
Payload	500g
Position repeatability	0.2mm
Material	Aluminium Alloy 6061
Controller	Arduino Mege2560
Commuication	UART/Bluetooth
Power Supply	12V 5A DC