Final Report

Computer Vision Assisted Robotic Arm

Shen Si Yuan

UID: 3035141542

2018-04-10

Abstract

Robotic arms have a significant amount of applications today. However, most of the robotic arms today are either operated by human or programed to perform a sequence of predefined motions. Therefore, the motivation of this project is to use computer vision technology to make a “smarter” robotic arm. The goal is to let the computer vision program guide the robotic arm to pick and place small objects. Approaches are divided in mainly two parts: hardware and software. And the same applies to the deliverables which includes robotic arm (hardware), computer vision program and an Arduino program.

Table of Content

List of Figures

List of Tables

1. INTRODUCTION

In this section, historical background and the prospects of the robotic arm will be introduced first. Then technical limitations of current models will follow. At the end of this section, it will conclude by summarizing project motivations.

1.1 History

Since electric and production line started to be adopted by factories during early 20th century, a significant amount of efforts has been made on improving production line efficiency. The first appearance of industrial robot was in 1937. Griffith P. Taylor, the inventor, built the crane-like machine which can perform five axes of movement. It was powered by a single electric motor and pre-programmed to stack wooden blocks in certain patterns. The very first arm solution of ***industrial*** robotics by Victor Scheinman at Stanford University, 1969, was an all-electric, 6-axis articulated robot which can accurately follow arbitrary paths in space. His design greatly widened the potential use of the robot to perform relatively sophisticated tasks, for example, assembly and welding. And IRB 6, which is believed to be the world's first *commercial* micro-processor controlled robot, was introduced by ABB Robotics (formerly ASEA) in 1973.

Europe and America have long been one of the global frontrunners in the race of automation manufacturing. However, the strongest growth drivers for the robotics industry are found in China. According to the 2016 World Robotics Report from International Federation of Robotics (IFR), by 2019, there will be 2.6 million units of robots deployed worldwide, which is one million units more than in the record-breaking year of 2015 **[1]**. (See **Figure 1**).

1.2 Technical Limitations

Despite the growing demand in robotic arms, most of the current models can only perform simple motions such as drilling, picking and placing. Their mechanical limitation is still a hot topic in engineering. In addition, majority of the *commercial* robotic arms are either manually controlled by operators or programmed to perform a sequence of predefined motions. Lack of independency and flexibility hinder the robots from massive use because they are usually unable to automatically handle more

complex situations with highly unpredictable variables such as outer space, abysmal sea, and even daily environment.

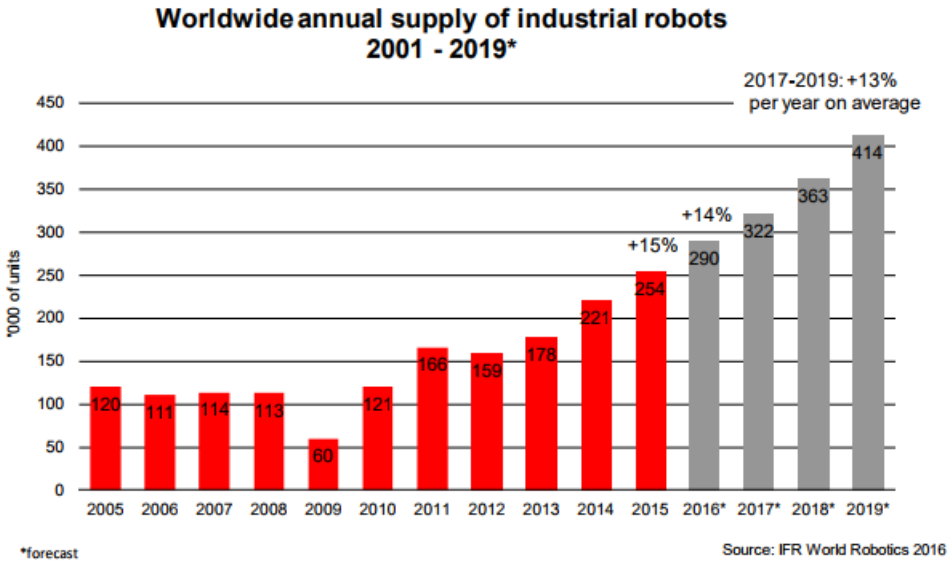

Figure 1

1.3 Motivation

The motivation is to overcome robotic arm's software limitations by using computer vision. This project aims to make flexible, highly automatic robotic arms which can recognize circles, plan their moves and pick the objects and put them together.

2. OBJECTIVES

The goal of the project is to enhance the functionality of robotic arms using computer vision. To be more specific, a robotic arm will be deployed to pick and place objects under the instruction of computer vision program.

To achieve this goal, it is necessary to complete several stages (in chronological order) during the project (see **Figure 2**). There are two main stages in the project, the first stage (arrows colored with green) is to make a controllable hardware and includes three sub-objectives (text in the green arrows): modeling, assembly and control. The second stage (arrows colored with blue) is to use computer vision to enhance the control software. There will be two modules (text in the blue arrows) that have different functions: vision and communication.

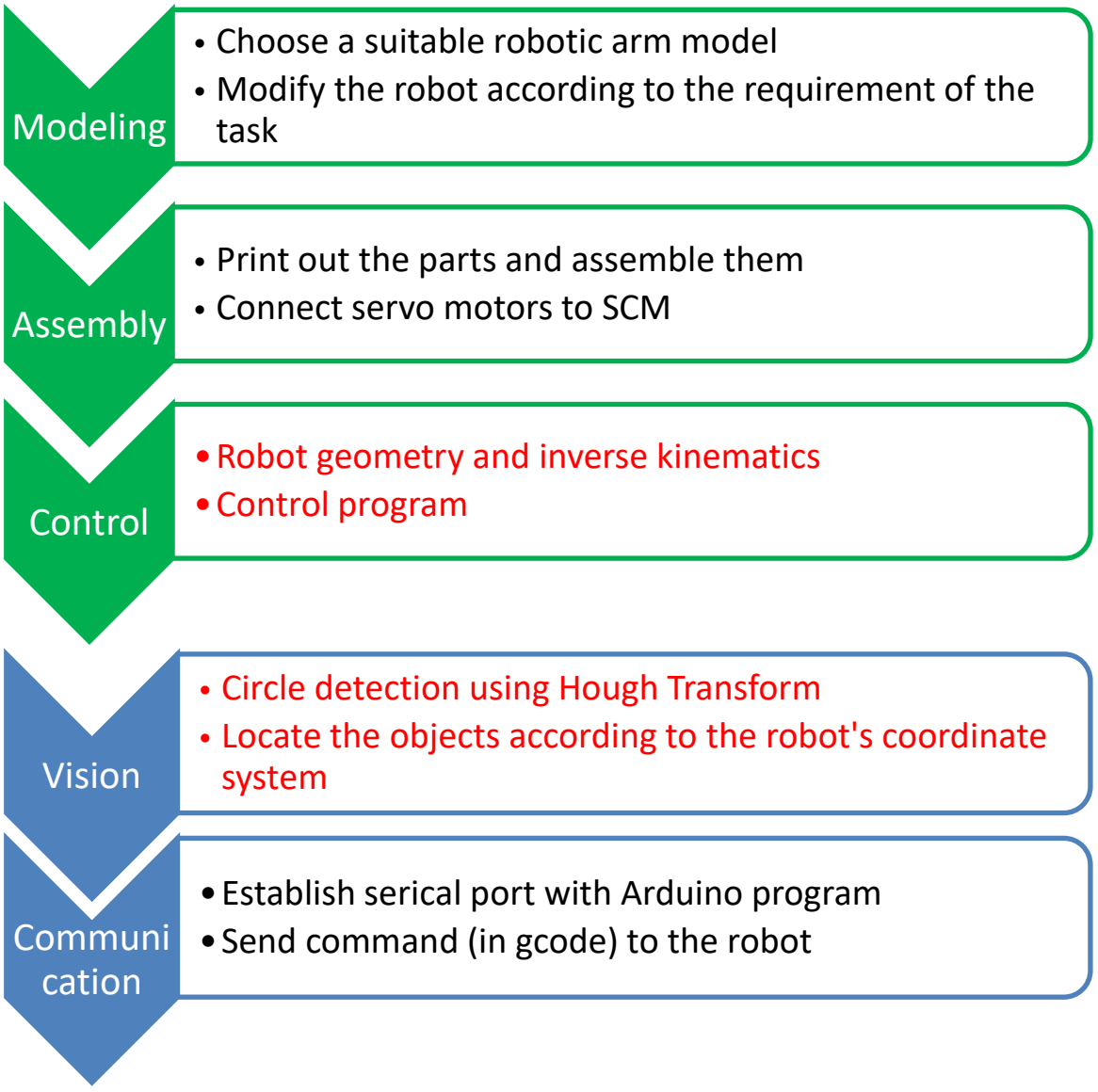

Figure 2, objectives

The core of this project is composed of two key sub-objectives (highlighted in red in **Figure 2**): control and vision. Control is the central motor structure of the robot, it provides low level control of the body. Vision is the eye of the robot, it percept the environment and process the visual information.

3. METHODOLOGY

This section is the expansion and explanation of the objectives showed in the previous section. The discussion of methodology follows the same order as **Figure 2**. Subsection **3.1** corresponds to the first stage of the project and subsection **3.2** corresponds to the second stage. Each subsection lists the criteria to justify its main engineering choices. Limitations will also be analyzed throughout each subsection respectively. A summary of methodology will be displayed at the end of this section.

3.1 First Stage: Hardware

The first stage is to make a hardware which is suitable for picking objects and develop the **low-level** control APIs. After the first stage, the robotic arm should have the following features:

1. Controllable: User can move the robotic arm on an electronic device using command or a GUI.
2. Suitable for the project's tasks: The robotic arm should be able to pick and place objects without any assist from the user other than controlling through a device.
3. Precise: The robotic arm should be able to reach a certain location precisely with less than an error of 5mm along each axis.
4. Fast: The robotic arm should be able to reach any location within its hardware limit in less than 4 seconds.

The first stage is divided into three objectives to achieve the goal.

3.1.1 Modeling

A suitable robotic arm model (see **Figure 3, Figure 4**) from *thingiverse.com* was chosen. The model satisfies the following criteria:

1. Easy to make: Main parts of the robotic arm are 3D-printable and the total number of parts is less than 80.
2. Moderate size: The geometry of the robotic arm (**Figure 4**) indicates that it is able to cover more than half of the chess board (28cm×28cm).
3. Driver's compatibility: The single-board microcontroller of the robotic arm: *Arduino Mega 2560* provides sufficient room and opportunities to maintain the simplicity and effectiveness of the *Arduino platform*[3]. In addition, the platform can run on Windows and has a mature, well-documented desktop IDE. Hence, it is suitable for any further expansion such as adding computer vision program.
4. Hardware performance: The robotic arm uses the common *NEMA 17 step motors* which provide enough power for fast movements and high precision.

However, the original short cuboid fingers lead to situations that the robot may not be able to hold a ball firmly. In addition, other objects may be touched and interfered while the robotic arm is picking. Therefore, modifications are needed to improve the suitability of the fingers for the task.

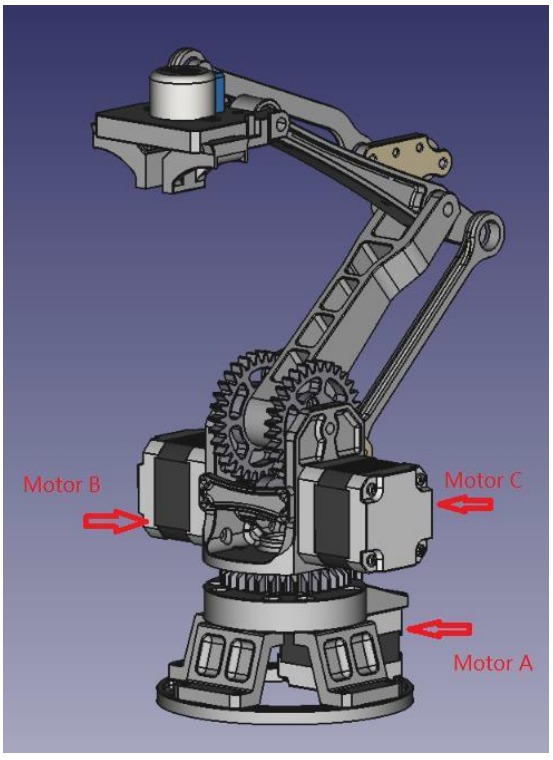

Figure 3, model of the robotic arm

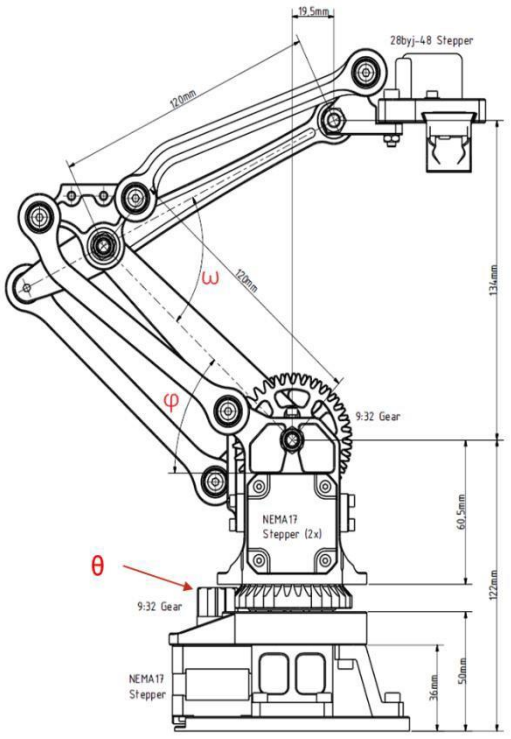

Figure 4, geometry of the robotic arm

The robotic arm in this project adopt the design in showed in **Figure 5** which improves the original version (**Figure 3**). The fingers are longer and have serrated edge to stable the chess piece.

Figure 5, modification

3.1.2 Assembly

Before the 3D-printing, model of the parts were converted to STereoLithography (.stl file format) and the printer parameters (printing temperature, wall thickness, density etc.) were set according to different requirements of the parts (strenghtness, weight etc). *Cura* converted the .stl file into the final G-code (.gcode file format) which can be recognized by the 3D printer. **Figure 6** shows the workflow of modeling and assembly. The blue textbox indicates the software/hardware used in each process respectively.

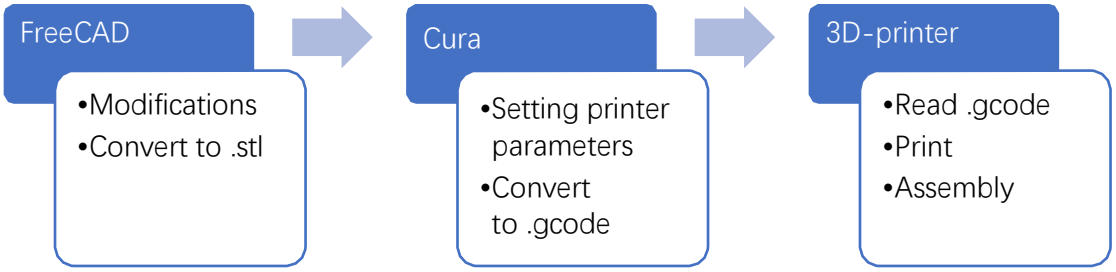

Figure 6, workflow of hardware assembly

3.1.3 Control

Picking and placing chess pieces require accurate positioning of the robotic arms. Hence, it is critical to analyze the robot geometry (see **Figure 4**). Inverse kinematics determine the higher/lower/base joint parameters ω/φ/θ (see **Figure 4**) in order to reach the desired position. It translates the motion plan which specifies the movement of the robotic arm to achieve the tasks into joint actuator (motor) trajectories for the robot. In this case, the desired position (interface input) will be represented by a set of coordinates in the robotic arm's coordinate system (**Figure 8**). The interface output is

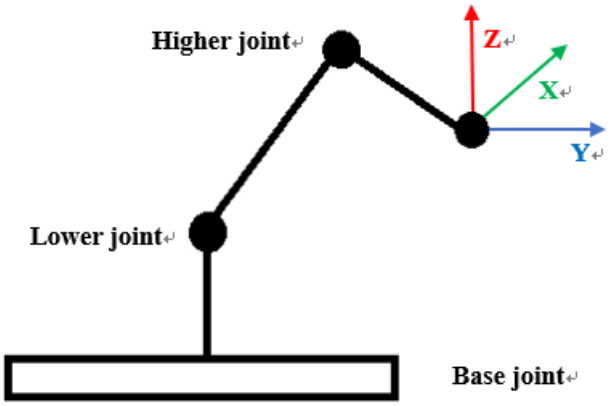

Figure 7, coordinate system of the robotic arm

the control flow of each motors (see **Figure 3**). **Motor A** controls the angle of the base joint, it will rotate the entire robotic arm. The lower and higher joints are controlled by **Motor B** and **C** independently. However, each increase/decrease in x/y/z coordinates may trigger all the three motors to work together.

Calculation details: Figure 8 is the view of the robot under three dimensional cartesian system. **Figure 9** is the simplified the model of the robotic arm's main geometry.

Parameters (known parameters are in boldface):

1. **Origin: (0, 0, 0) at base joint** **Figure 9**
2. **x, y, z: the coordinate of the robotic arm** **Figure 8**

   x, y, z is known in a given gcode command, e.g. G1 X0 Y120 Y120
3. **Length of the arm is fixed: 120mm & 120mm** **Figure 9**
4. Rrot: the radius of (x, y, z) projected in the x-y plane **Figure 8&9**

   $Rrot = (x^2 + y^2)^{1/2}$

5. rot: rotation angle of the base joint **Figure 8**

   $rot = \sin^{-1}(x/Rrot)$

6. Rside: the length from base joint (origin) to (x, y, z) **Figure 8&9**

   $Rside = (x^2 + y^2 + z^2)^{1/2}$

7. high: rotation angle of the higher joint **Figure 8&9**

   $high/2 = \sin^{-1}(0.5Rside/120mm)$

   Thus, $high = 2\cos^{-1}(Rside/(2*120mm))$

8. $\alpha = \pi/2 - high/2$ **Figure 9**

9. $\omega = [\cos^{-1}(Rrot/Rside)] * (z/|z|)$ **Figure 9**

10. low: rotation angle of the lower joint (it can be negative) **Figure 8&9**

    $low = \pi/2 - (\alpha + \omega)$

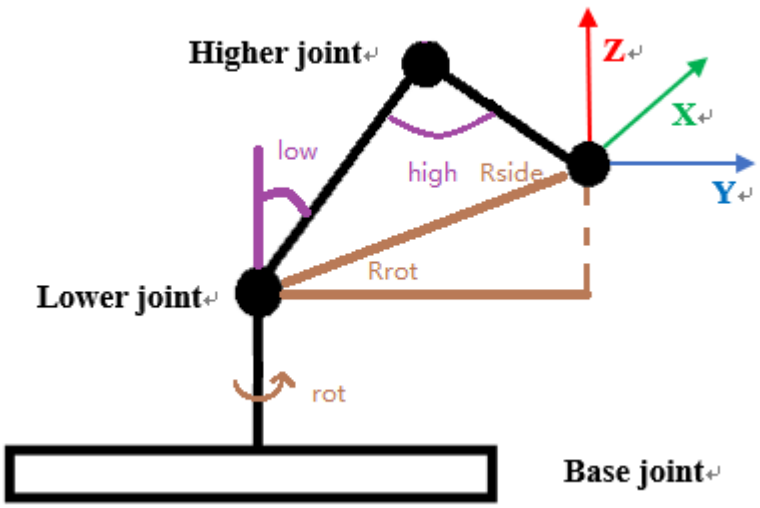

Figure 8, 3-D cartesian view

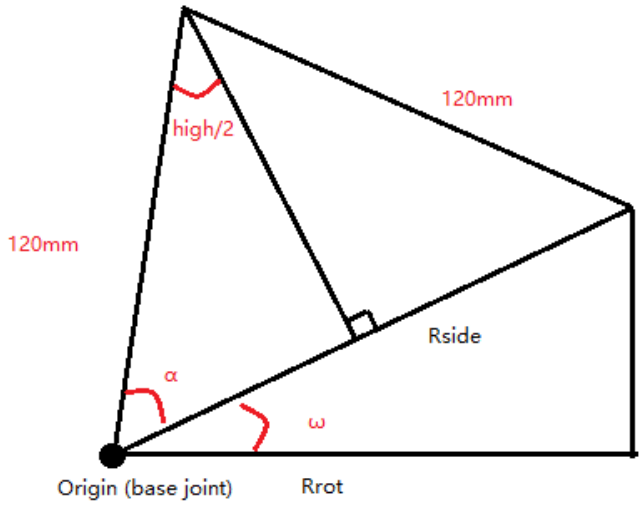

Figure 9, simplified 2-d side view

The calculation process has been implemented into the basic control program. The program was uploaded to the microcontroller using the Arduino platform. User can control the robotic arm using a simple GUI (see **Figure 10**). The following list describes the function of each area in the corresponding labeled red box.

1. Hardware connection
2. Current position of the robotic arm
3. Hardware control dashboard
4. Arm control panel
5. Command box
6. Command log window (show the command history)Robotic arm log window (show the position history of the arm)

Hardware control dashboard is used to turn on/off motors and adjust their step length, it also controls the fingers.

The robotic arm can be controlled to move along each axis X, Y and Z by using the control panel. User can also choose to directly type in the target location in the command box. For example, command ‘G1 X0 Y120 Z120’ asks the robot to go to (0, 120, 120) with respect to its own coordinate system.

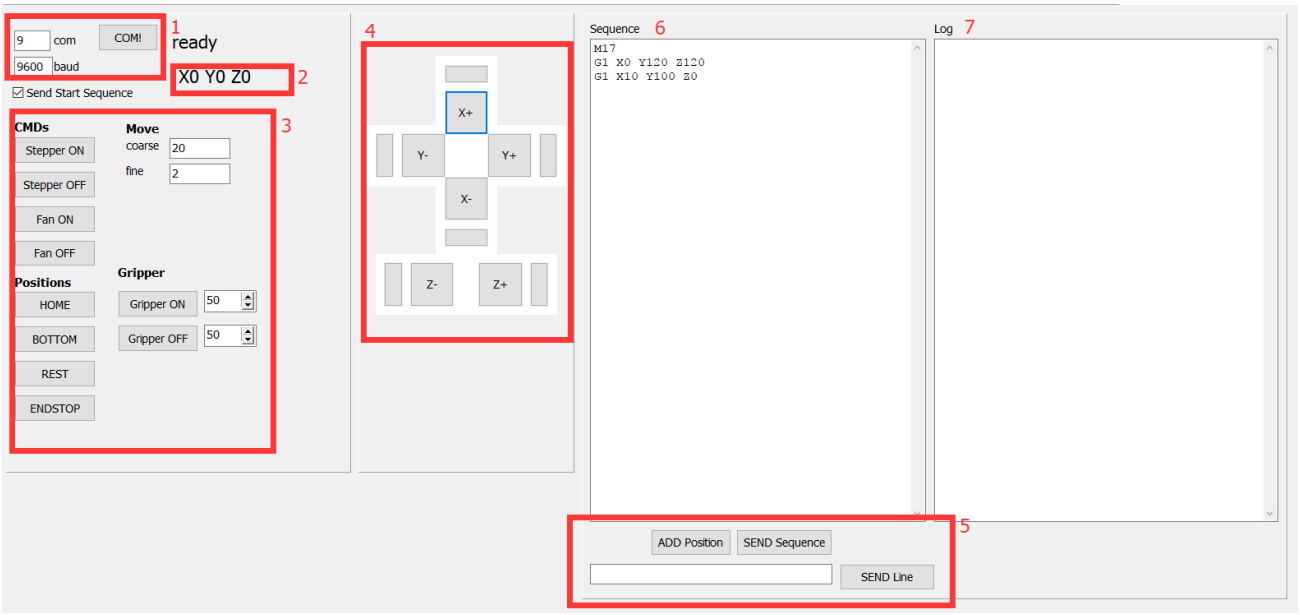

Figure 10, GUI of the control program

3.2 Second Stage: Software

With the control program, we can move the robotic arm manually. However, like most commercial robotic arms today, it still relies on human to perform tasks. Therefore, the second stage aims to develop programs that can make the robotic arm pick and place the objects automatically. **Figure 11** shows the relation ship between the two programs: vision and control.

The vision program detects all the circles in camera's view, collect their information (coordinate, radius) and generate command which tell the control program to fetch the objects. Two programs communicate through a serial port. Vision program use gcode command to guide the robotic arm and receive the feedback from Arduino.

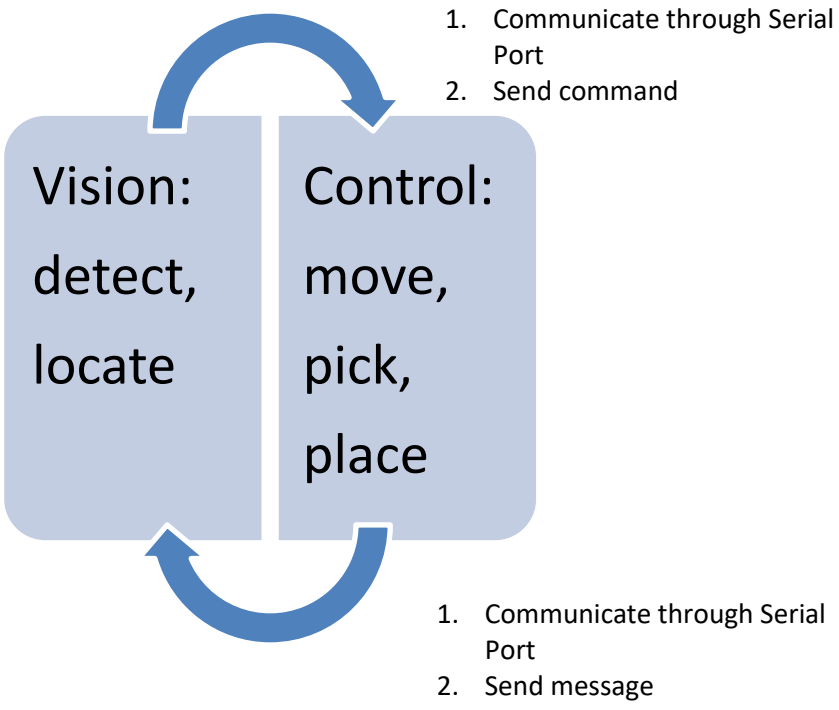

Figure 11, relationship of the two programs

3.2.1 Vision

As discussed in the previous section, vision program is one of the key processes of the whole project. It is the eye of the robotic arm. An simple RGB camera shall be used.

The vision program should have the following functions:

1. Recognize each circle in its view
2. Locate the circles
3. Transform the coordinates of the circles to robot's coordinate system
4. Generate command according to their locations
5. Update information when control program sends back messages
6. Repeat 1-5

Circle Detection, Circle Hough Transform (CHT) and Hough Gradient Method:

Hough transform is a feature extraction algorithm for finding regular shapes (line, circles etc.) in an image. The basic of Circle Hough Transform is that in a two-dimensional space, a circle can be described by:

$$(x^2 - a^2) + (y^2 - b^2) = r^2 \qquad (1)$$

Where r is the circle's radius and (a, b) is its center. Given a fixed (x, y), this equation provides a surface on the three-dimensional parameter space (a, b, r). The parameters can be identified by intersecting many theses surfaces defined by the points on the 2D circle.

For each point (x, y) on the original circle, it can define another circle centered at (x, y) with radius r. The intersection point of all such circles in the parameter space would be the corresponding true center of the original circle. CRH calculate the accumulator matrix (3D parameter space) and choose the point with maximum voting. The whole process will be iterated through all possible radius.

However, one of the greatest drawback of the traditional CRH is that a three-dimension accumulator requires much more memories and run in much slower speed. Hough Gradient Method avoid this problem by using a somewhat trickier way and it works as follows. First the image is passed through an edge detection phase (canny edge detector). Next, for every nonzero point in the edge image, the local gradient is considered. Using this gradient, every point along the line indicated by this slope – from a specified minimum to a specified maximum distance – is incremented in the

accumulator. Ah the same time, the location of every one of these nonzero pixels in the edge image is noted. The candidate centers are then selected from those points in the two-dimensional accumulator that are both above some given threshold and larger than all their immediate neighbors. These candidate centers are sorted in descending order of their accumulator values, so that the centers with the most supporting pixels appear first. Next, for each center, all of the nonzero pixels are considered. These pixels are sorted according to their distance from the center. Working out from the smallest distances to the maximum radius, a single radius is selected that is best supported by the nonzero pixels. A center is kept if it has sufficient support from the nonzero pixels in the edge image and if it is a sufficient distance from any previously selected center.

This implementation enables the algorithm to run much faster and, more importantly, helps overcome the problem of the otherwise sparse population of a three-dimensional accumulator, which would lead to a lot of noise and render the results unstable.

On the other hand, this algorithm has several shortcomings that should be aware of.

First, the use of the Sobel derivatives to compute the local gradient is not a numerically stable proposition. It is expected to generate some noise in the output.

Second, the entire set of nonzero pixels in the edge image is considered for every candidate center. Hence, if the accumulator threshold is too low, the algorithm will take a long time to run.

Third, because only one circle is selected for every center, if there are concentric circles then it will detect only one of them.

Finally, because centers are considered in ascending order of their associated accumulator value and because new centers are not kept if they are too close to previously accepted centers, there is a bias toward keeping the larger circles when multiple circles are concentric or approximately concentric.

3.2.2 Coordinate System Transformation

If the control program provides a controllable body and the vision program provides the eye, then the coordinate system transformation is the brain of the robotic arm which takes in visual information, process data and give commands to the body accordingly. It has the following functions:

1. Calculate transformation matrices between three coordinate systems: camera and robotic arm
2. Indentify the changes of real-world status using the information provides by the vision program
3. Generate gcode command (e.g. G1 X30 Y40 Z50) according to the changes of status
4. Take in feedback messages from Arduino board and update status

To give command that will guide the robotic arm to pick and place the objects correctly, first we need to accurately transform the (x, y) coordinate and the radius given by the Hough Transform algorithm to robotic arms coordinate system.

In order to reduce the requirements of the hardware as well as speed up the programs. The vision program abandoned the previously used RGBd camera and use regular web camera instead. However, the cost is loss of depth information of each pixel. Hence, the environment needs to be fixed in order to recover the three-dimensional information of each object.

The program implements the transformation in the most simplified condition: The view plane of the web camera is parallel to the table (base of all the objects) and the distance between the camera and the table is fixed and known.

Figure 12 illustrate the preset environment.

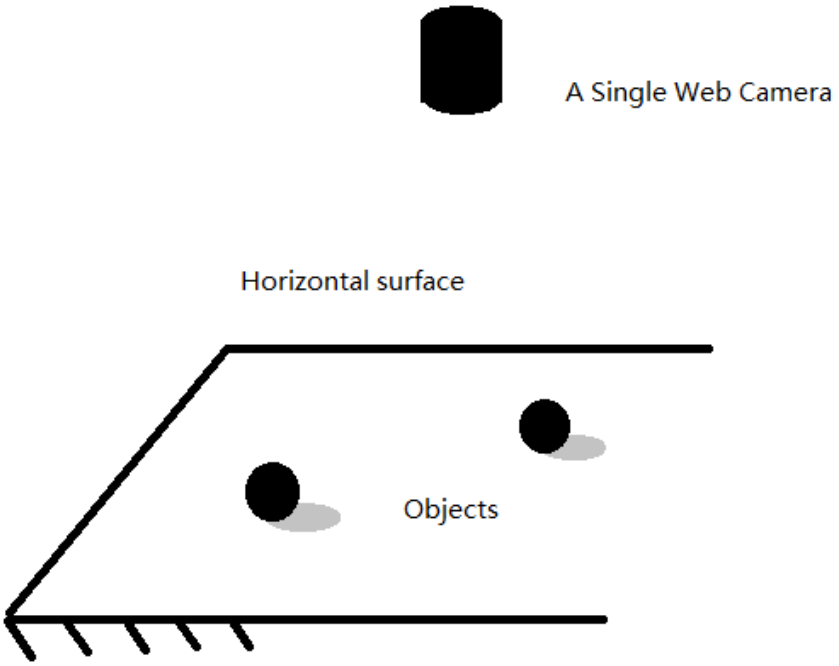

Figure 12, preset environment

Under this setting, the transformation between a pixel (p, q) and the robotic arm's coordinate system (x, y, z) can be represented as:

$$(x, y, z) = (\lambda p cos\theta - \lambda q sin\theta - b1, \lambda q cos\theta + \lambda p sin\theta - b2, r + b3)$$

Where λ is the scaling factor owning to the projection from the surface to the view plane, θ is the angle between the two coordinate systems and $b1, b2$ are the biases between the two origins and $b3$ serve as the a constant for the robotic arm to pick the objects properly.

3.2.3 Serial Communication and Command

Communication Protocols: Table 1 describe the communication protocols (commands) from the vision program to the control program. And **Table 2** describe the communication protocols (messages) from the control program to the vision program.

Format	**Description**
String “G0 X%d Y%d Z%d”	Move XYZ in mm (cartesian). Always uses Absolute coordinates. On every move there is an acceleration and deceleration.
String “G1 X%d Y%d Z%d”	Same as G0
String “G4 T%d”	Dwell / Sleep T in milliseconds
String “M3 T%d”	Close Gripper / Aux Motor in T steps. Cannot move simultaneously with ‘G’ command.
String “M5 T%d”	Open Gripper / Aux Motor in T steps. Cannot move simultaneously with ‘G’ command.
String “M17”	Enable Stepper Motors
String “M18”	Disable Stepper Motors

Table 1, commands

Format	**Description**
Char ‘C’	Current (set of) commands completed, request for further commands.
Char ‘E’	Error occurred during execution of command.

Table 2, messages

Pseudo Code for Vision Program:

……

While(**flag**):

frame ← StreamFromWebCamera()

pframe ← PreprocessFrame(**frame**)

circles ← HoughCircle(**pframe**)

for each **circle** in **circles**:

robot_coordinate ← TransformCoordinate(**circle**)

Append to **circles_need_picking**

for each **circle_need_picking** in **circles_need_picking**

command ← GenerateCommand(**circle_need_picking**)

Serial.SendCommand(**command**)

message ← Serial.ReceiveMessage()

if (**message** == “C”): continue

4. Results

In this section, the result of the projected will be displayed. It will follow the workflow of the system (**Figure 13**).

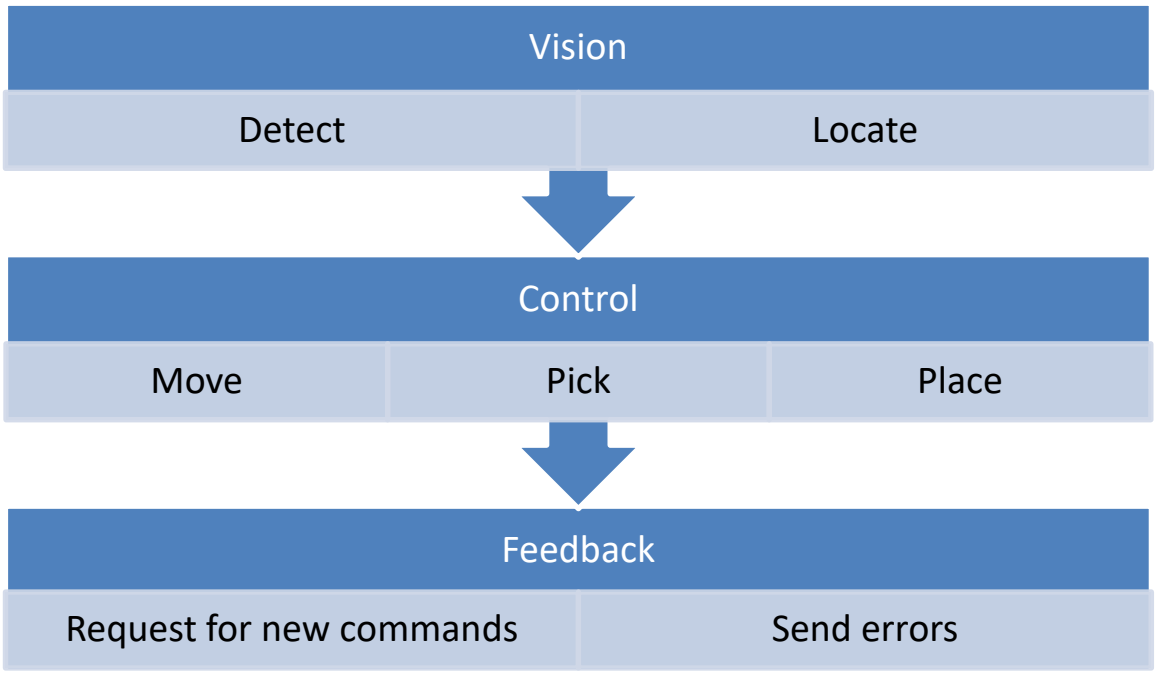

Figure 13, workflow

4.1 Environment settings

To run the program and make the computer vision algorithm works efficiently, the environment needs:

1. Indoor with appropriate lighting conditions
2. A smooth, stable and horizontal surface (e.g. table)
3. A background with color that contrasts the objects (e.g. white objects and black background)
4. Objects need to be of similar height and size
5. The distances between objects cannot be too small, otherwise the robotic arm may not be able to pick them properly.

4.2 Vision outcome

Figure X is an image generated by the vision program showing the circles (three white balls and three green cylinders) marked by Hough Transform algorithm.

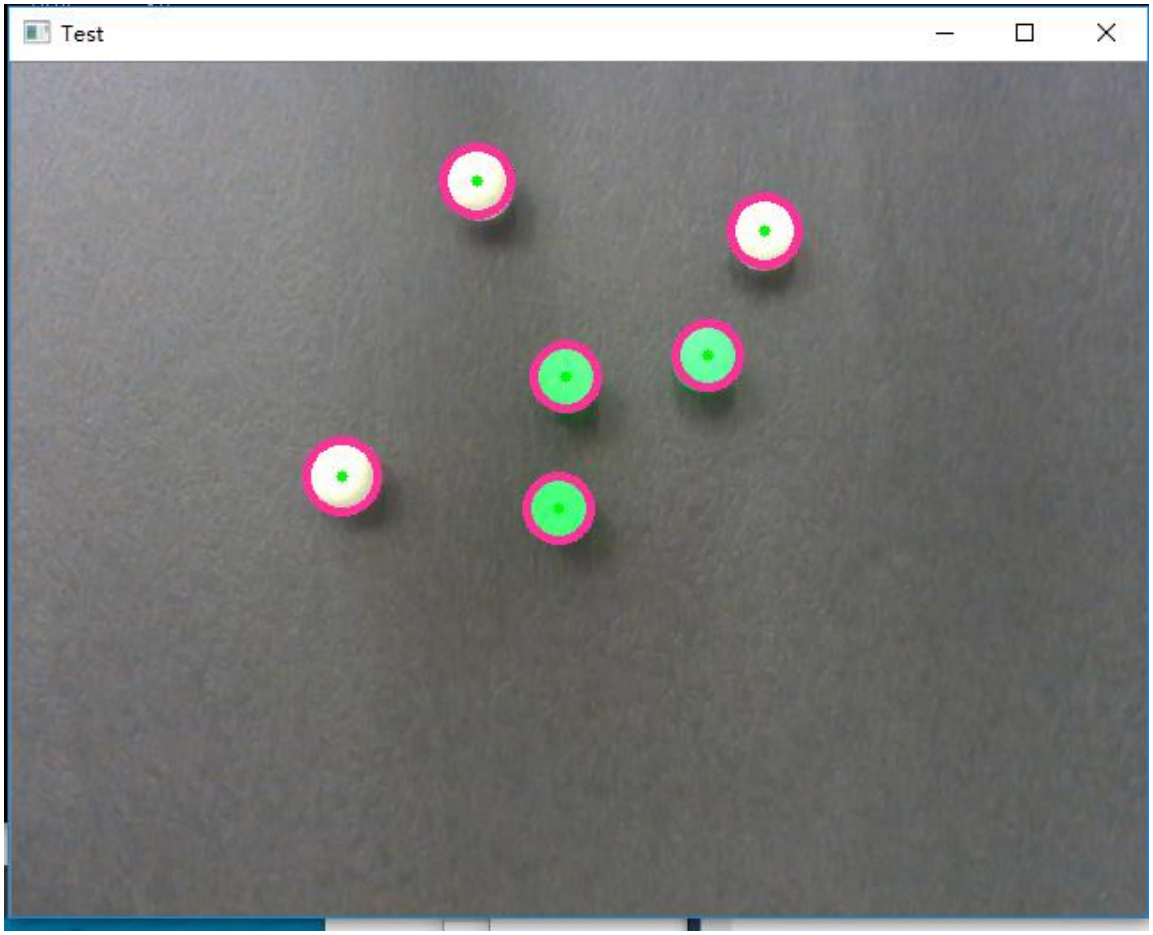

Figure 14, vision outcome 1

Figure 15 is the console output which indicates the circle's positions (on the view plan) and their radius.

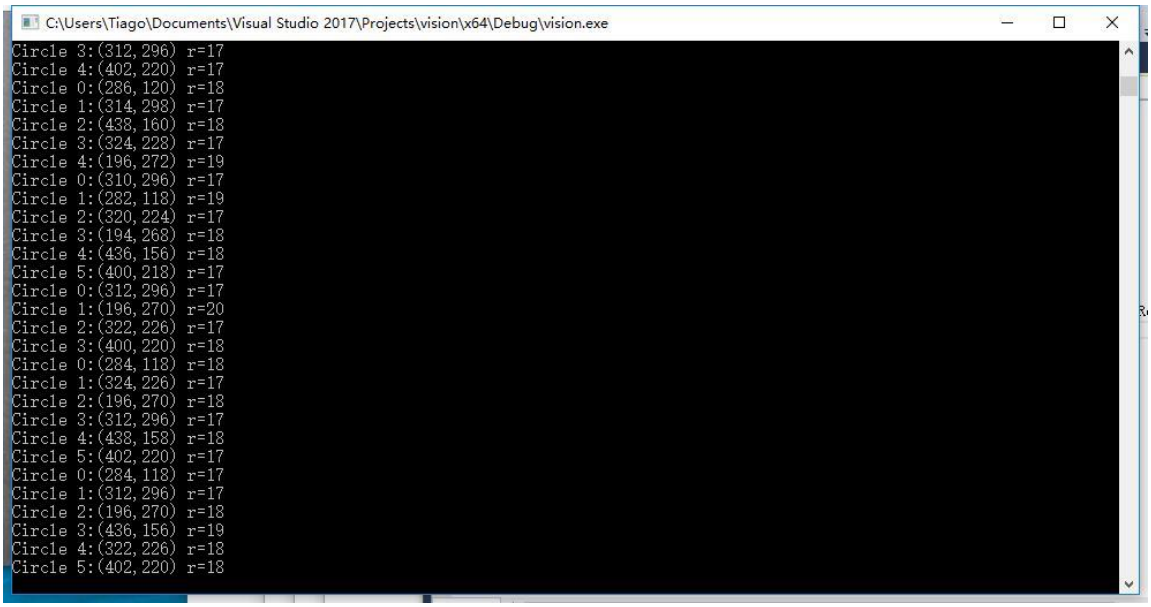

Figure 15, vision outcome 2

As shown above, there are totally six circles in the environment. However, the Hough Transform did not recognize all of them each time it is called. One of the main reason is that the parameters of Hough Transform may not be perfectly set. But in the long-run, it will detect all the objects of interests. In addition, the program seldom has false positive result, which means it wrongly detect and locate a circle (no such case in this environment setting so far). Thus, it has minimum effect on the picking.

In addition, if the picking fails, for example, if the object falls when the robotic arm is trying to pick it. Although the robotic arm will no longer pick it during current iteration. The object will be detected and located at the beginning of the next iteration. It is also a demonstration of the flexibility computer vision can provide for the robotic system.

4.3 Final Result

A demo is provided which record two iterations (as described in the pseudo code).

5. Conclusions and Future Works

5.1 Conclusions

With the help of computer vision, the robotic arm can automatically pick and place balls and cylinders and has a moderate tolerance to failures (explained in the previous section **4.1 Vision outcome**), thus, the goal of this project is fulfilled.

5.2 Future Works

Owning to capacity and time limit, there are still many spaces for improvement in this project:

1. The vision program can only detect circles.
2. The vision program takes sample frame every few seconds. Hence the robotic arm can only pick objects that are not moving.
3. The system is not robust to changes of environment (lighting condition, background color, camera position etc.).

Thanks to the booming research in computer vision and neural network, computer now achieves a lower error rate in object recognition than average human performance. If combined with neural network, the vision program will be more powerful and the whole system will be more flexible.

References:

All the references of this project are listed below

1. 2016 World Robotics Report, *International Federation of Robotics (IFR), 2016.*
2. http://cs231n.github.io/classification/. Retrieved at 2017-11-21.
3. https://www.arduino.cc/en/Guide/ArduinoMega2560. Retrieved at 2017-11-21
4. Robotic Arm, https://www.thingiverse.com/
5. RAMPS 1.4, http://www.reprap.org/wiki/RAMPS_1.4
6. Cura, https://ultimaker.com/en/products/ultimaker-cura-software
7. FreeCAD, https://www.freecadweb.org/
8. OpenCV Library, https://opencv.org/
9. Circle Hough Transform, https://en.wikipedia.org/wiki/Circle_Hough_Transform
10. Learning OpenCV, Retrieved from http://www-cs.ccny.cuny.edu/~wolberg/capstone/opencv/LearningOpenCV.pdf
11. OpenCV Circle Hough Transform, https://blog.csdn.net/qq_15947787/article/details/50802953
12. Ball Tracking with Hough Transform, https://blog.csdn.net/xiao__run/article/details/76660362
13. Arduino and C++ (for Windows), https://playground.arduino.cc/Interfacing/CPPWindows