UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF ELECTRONICS, TELECOMMUNICATIONS AND INFORMATION TECHNOLOGY

AUTONOMOUS SYSTEM FOR PERFORMING DEXTEROUS, HUMAN-LEVEL MANIPULATION TASKS AS RESPONSE TO EXTERNAL STIMULI IN REAL TIME

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the Degree of Engineer in the domain Technology and Telecommunication Systems
Study program: Telecommunications and Information Technologies

Thesis advisor(s):
Prof. Corneliu BURILEANU, Ph. D.
Associate Prof. Horia CUCU, Ph. D.

Student:
Ana-Antonia NEACȘU

Bucharest
2017

Annex 1

University "Politehnica" of Bucharest
Faculty of Electronics, Telecommunications and Information Technology
Department : Telecommunications

Department Director's Approval *:

Conf. Eduard POPOVICI, Ph D.

[illegible]

DIPLOMA THESIS
of student NEACŞU M. Ana-Antonia, 441G

1. Thesis title: **Autonomous system for performing dexterous, human-level manipulation tasks as response to external stimuli in real time**

2. The student's original contribution will consist of (not including the documentation part):

The purpose of this thesis is developing an autonomous system that controls a robotic arm, which performs human-level actions based on external stimuli, represented by real time images from a Kinect camera. The system will be solving a complex puzzle, namely a Pyramix puzzle (Rubik's pyramid), demonstrating the degree of movement complexity that the Kinova robotic arm can achive. The system is composed of three important parts: the first's one main purpose is to capture real time images from the Kinect sensor and processes them into input data for the second module. The second part, the core of the system, performs all necessary computations in order to make a movement decision based on the available data. The third part represents an interface with the robotic arm, transposing the decision from the second block into pure movement data, passed to the Kinova's controller.

3. The project is based on knowledge mainly from the following 3-4 courses: Microcontrollers, Microprocessors architecture, Object-Oriented Programming

4. The Intellectual Property upon the project belongs to*: Student, Speech&Dialogue Research Laboratory*

5. The research is performed at the following location: *Speech&Dialogue Research Laboratory*

6. The hardware part of the project stays in the property of : *Speech&Dialogue Research Laboratory*

7. The thesis project was issued at the date: 28.07.2016

Thesis Advisor:

Prof. Corneliu BURILEANU, Ph. D.

[illegible]

Lect. Horia CUCU, Ph. D.

[illegible]

STUDENT:

Ana-Antonia NEACŞU

[illegible]

STATEMENT OF ACADEMIC HONESTY

I hereby declare that the thesis "*Autonomous System For Performing Dexterous, Human-Level Manipulation Tasks As Response To External Stimuli In Real Time*", submitted to the Faculty of Electronics, Telecommunications and Information Technology in partial fulfillment of the requirements for the degree of Engineer of Science in the domain Technology and Telecommunication Systems, study program Telecommunications and Information Technologies, is written by myself and was never submitted to any other faculty or higher learning institution in Romania or any other country.

I declare that all information sources I used, including the ones I found on the Internet, are properly cited in the thesis as bibliographical references. Text fragments cited "as is" or translated from other languages are written between quotes and are referenced to the source. Reformulation using different words of a certain text is also properly referenced. I understand plagiarism constitutes an offence punishable by law.

I declare that all the results I present as coming from simulations and measurements I performed, together with the procedures used to obtain them, are real and indeed come from the respective simulations and measurements. I understand that data faking is an offence punishable according to the University regulations.

Bucharest, *July 2017*

Ana – Antonia **NEACȘU**

[illegible]

(Student's signature)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AC – Alternative Current

API – Application Programming Interface

CAD – Computer-aided design

DC – Direct Current

FPS – Frames per second

FTDI – Future Technology Devices International

GND – Ground

HSV – Hue Saturation Value

ICSP – In Circuit Serial Programming

IR - Infrared

L4E – Last Four Edges

LED – Light-emitting Diode

LL – Last layer

MISO – Master In Slave Out

MOSI – Master Out Slave In

NUI – Natural User Interface

OKA – Oriented Keyhole Algorithm

PWM – Pulse Width Modulation

RGB – Red Green Blue

SCK – Serial Clock

SD – Standard Deviation

SDK – Software Development Kit

SPI – Serial Peripheral Interface Bus

SS – Slave Select

USB – Universal Serial Bus

ZIF – Zero Insertion Force

CHAPTER 1
INTRODUCTION

1.1 THESIS MOTIVATION

Nowadays, society tries to accept and integrate persons with various disabilities. They comprise an estimated population of one billion people globally, of whom eighty percent live in developing countries and are overrepresented among those living in absolute poverty. A first step in the integration process is to find a way to improve the quality of their life.

The core of this project is the robotic arm created by Kinova robotics to aid people who are battling disabilities. For someone with severe motion disability some trivial tasks, such as picking up a glass of water or opening a door may represent a great challenge. In this context, it is imposed to find an efficient solution to give these people some degree of independence. Hence, the need to develop a system capable of performing human level motions with as less outside intervention as possible.

This system has the purpose of helping people with different types of conditions: traumatic injuries, such as spinal cord injuries, lost or damaged limbs, diseases and Congenital Conditions, like Cerebral Palsy, Muscular Dystrophy, Multiple Sclerosis, Spina Bifida, ALS (Lou Gehrig's Disease), Arthritis, Parkinson's disease, Essential Tremor etc.

1.2 MAIN OBJECTIVE

In this context, this thesis aims to create an autonomous system, capable to perform a given task without any kind of human intervention, with human-like dexterity, as response to external stimuli, in real time.

To prove this point, this project implies a system that solves automatically, a complex puzzle, namely a Pyraminx puzzle (Rubik's pyramid) using the robotic arm developed by Kinova Robotics. The system is composed of three important parts: the first's one main purpose is to capture real time images from the Kinect sensor and to process them into input data for the second module. The second part, the core of the system, performs all necessary computations in order to make a movement decision based on the available data. The third part represents an interface with the robotic arm, transposing the decision from the second block into pure movement data, passed to Kinova's controller.

The steps required to complete these objectives are listed below:

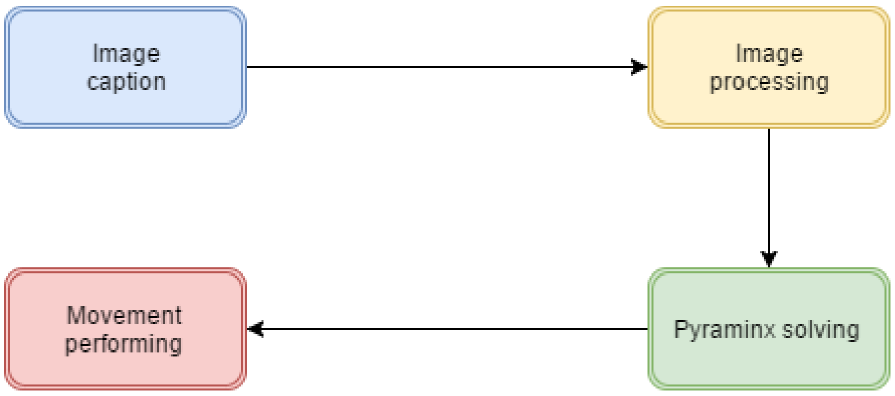


Figure 1.1 Implementation steps

1.3 SPECIFIC OBJECTIVES

- Developing an algorithm that solves the Pyraminx using the robotic arm – All the existing methods for solving such puzzles involve the use of both hands. Since I only have access to one robotic hand, I create my own custom method that can be performed by the robot. My purpose is to completely solve this puzzle using a robot, not necessary in the optimal way possible or in the shortest period.
- Programming the Robot to automatically solve the Pyraminx – Using data received from the algorithm, the arm performs the necessary actions to solve the puzzle
- Creating a stand for the pyramid – the puzzle must stay in a position so that the robot can reach and rotate its pieces.
- Capturing images with a Kinect module – Images represent the real stimuli, and based on them the algorithm will find the solution of the puzzle.
- Implementing solution to get images of all the faces of the pyramid without manually moving camera to different angles – My main objective is to create an autonomous system without any human intervention
- Finding a method to determine the color of every piece of the puzzle – The solution of the puzzle implies that all the pieces from a certain face match in color, so the way the colors are arranged represent the input data for the solving algorithm.

The thesis is organized in six chapters, as follows:

Chapter 1 presents the motivation, the objectives and the outline of this thesis. In *Chapter 2* are detailed the hardware and software technologies used to develop this project. *Chapter 3* is the first chapter that illustrates contributions of the author of the thesis. It describes the data acquisition process and the image processing methods I have approached. *Chapter 4* deals with the development of an algorithm that solves the Pyraminx puzzle, explaining each required step. In *Chapter 5* focuses on the implementation the actual moves of the robotic arm. Finally, *Chapter 6* summarizes the main conclusions of the thesis and underlines the author's contributions.

CHAPTER 2 SOFTWARE TECHNOLOGIES

2.1 KINOVA – JACO^2 ROBOTIC ARM

Kinova designs and manufactures robotics platforms and components that are simple, efficient and safe under two business units: Assistive Robotics which empowers people with disabilities to push beyond their current boundaries and limitations while Service Robotics empowers people in industry to interact with their environment more efficiently and safely. [1]

For this project, I use the Jaco^2 model. Launched in 2010, Jaco is a six-axis robotic manipulator arm with a three-fingered hand. This robot significantly improves the lives of persons with reduced mobility by assisting anyone with an upper body mobility impairment to perform complex actions. The assistive robot was immediately adopted by many of those with upper body disabilities and Jaco soon made a name for itself. The features of the robot are listed below:

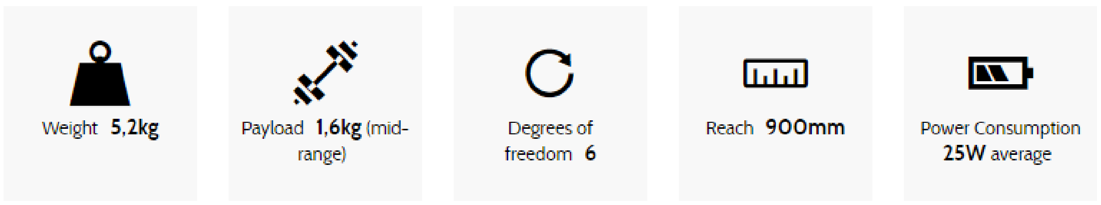

Weight **5,2kg**
Payload **1,6kg** (mid-range)
Degrees of freedom **6**
Reach **900mm**
Power Consumption **25W** average

Figure 2.1 Jaco Features [1]

- 6 movements in total
- Carbon fiber structure
- Lightweight
- Weather resistant
- Reach the floor with standard installation on wheelchair
- Option of use 2 or 3 fingers
- High friction rubber pads make grasping objects easier
- Optimized for activities of daily living
- Flexible fingers
- Adaptability to shape and size
- Current sensors and limitation

The arm can be controlled with the help of a joystick, but it can be also programmed, using an SDK provided by the manufacturer, for C++ programming language.

2.1.1 The arm

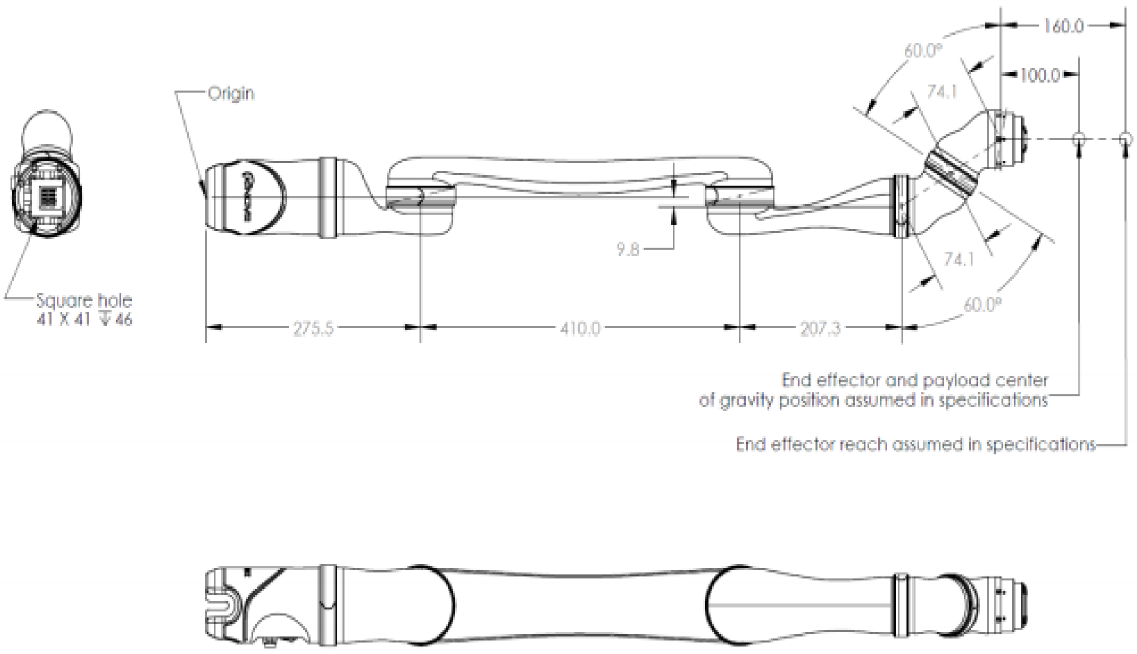

Origin
Square hole
41 X 41 ⩡ 46
160.0
100.0
60.0°
74.1
9.8
74.1
60.0°
275.5
410.0
207.3
End effector and payload center
of gravity position assumed in specifications
End effector reach assumed in specifications

Figure 2.2 Jaco – Arm specifications [2]

Table 2.1 Jaco^2 - Arm Specifications [2]

Specifications	
Total weight	**4,4 Kg**
Materials	Carbon fiber (links), Aluminum (actuators)
Payload	2.6 Kg (mid-range continuous payload capabilities) 2.2 Kg (full-reach peak/temporary payload capabilities)
Reach	90 cm
Joint Range After Start-Up (Software Limitation)	±27.7 turns
Maximum Linear Arm Speed	20 cm/s
Power Supply Voltage	18 to 29 VDC
Average Power	25 W (5W in STANDBY)
Peak Power	100 W
Communication Protocol	RS485
Communication Cables	20 pins flat flex cable
2 Expansion Pins on Communication Bus	yes
Water Resistance	IPX2
Operating Temperature	-10 °C to 40 °C expected

Jaco^2 enables users to interact with their environment with complete safety, freedom, and effectiveness. The arm moves smoothly and silently with unlimited rotation on each axis.

The axes are aluminum compact actuator discs (CADs) of a unique design. Each Jaco^2 robot arm consists of 2 distinct sets of 3 identical, interchangeable, and easy-to-replace CADs linked together by a ZIF (zero insertion force) cable.

Its main structure, entirely made of carbon fiber, delivers optimal robustness and durability as well as a cutting-edge look-and-feel. The arm is mounted on a standard aluminum extruded support structure that can be affixed to almost any surface.

Each actuator has a wide range of degrees that they can move freely, the motors having continuous rotation. The only limitations come from the architecture of the hand (some actuators cannot rotate completely). The values are listed in the table below:

Table 2.2 Jaco – Actuators and Fingers Specifications [2]

Actuators and Fingers Specifications		
	Minimum Position	**Maximum Position**
Actuator 1	-1000°	-1000°
Actuator 2	-1000°	-1000°
Actuator 3	-1000°	-1000°
Actuator 4	-1000°	-1000°
Actuator 5	-1000°	-1000°
Actuator 6	-1000°	-1000°
Finger 1	posSwitch	posSwitch + 60,010
Finger 2	posSwitch	posSwitch + 60,0
Finger 3	posSwitch	posSwitch + 60,0

2.1.2 *The gripper*

Table 2.3 Jaco K 3 - Gripper Specifications [2]

Gripper Specifications	
Fingers Quantity	**3**
Actuation System	Under Actuated
Actuators	One Per Finger
Actuators Sensors	Current
	Temperature
	Rotational Encoder
OPENING (Fingertip)	175 Mm
Min Object Diameter for Cylindrical Grip	45 Mm
Max Object Diameter for Cylindrical Grip	100 Mm
MIN OBJECT DIAMETER FOR OBJECT-ON-THE-GROUND Pinch	8 Mm
Total Weight	727 G
Gripping Force	
3 Fingers	40 N
2 Fingers	25 N
Opening or Closing Travel Time	1.2 Sec
Operating Temperature	-10 °C To 40 °C

The gripper consists of 2 or 3 underactuated fingers that can be individually controlled. Their unique bi-injected plastic structure endows them with great flexibility and unrivalled grip. In contrast to the previous version, Jaco² has a slimmer gripper and a friction pad which allows the fingers to adjust to any object whatever its shape; thus, they can gently pick up an egg, or firmly grasp a jar.

2.1.3 The controller

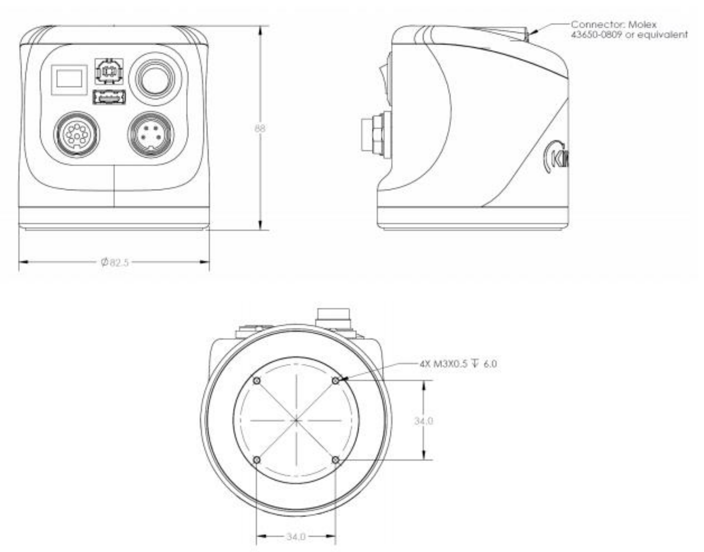


Figure 2.3 Jaco – Controller Specifications [2]

Jaco² can be controlled with a computer (see the "Software" section below) or Kinova's 3-axis, 7-button joystick. The control is intuitive and allows users to navigate using 3 different modes: translate, rotate and grip. It offers two types of control: Angular control and Cartesian control. The first one implies moving each actuator separately as shown in the image bellow:

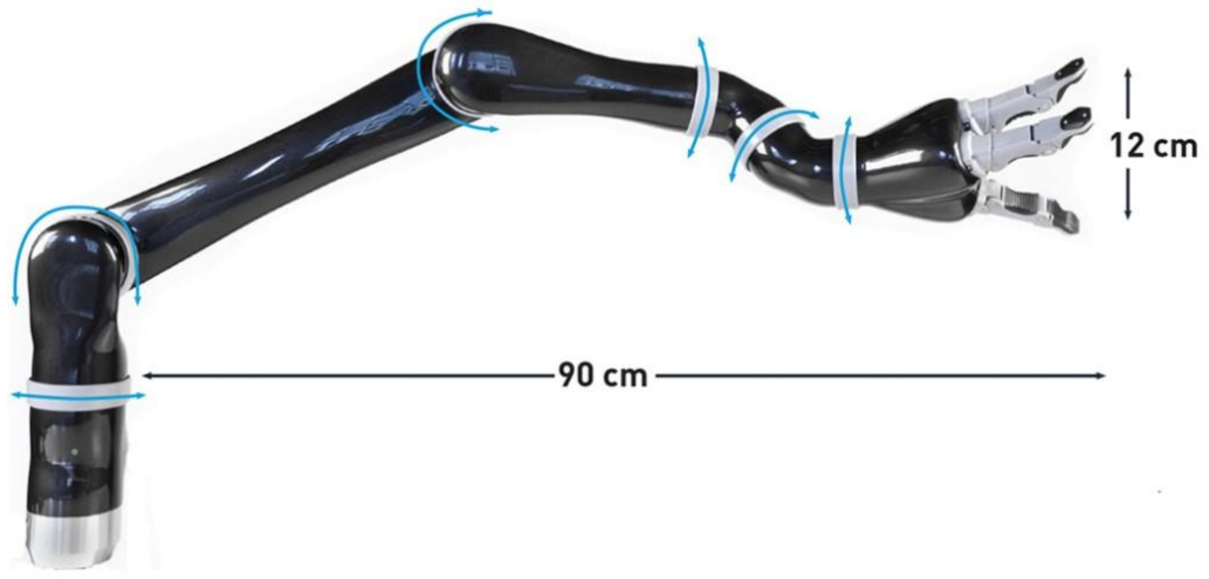


Figure 2.4 Angular Mode [1]

Cartesian Mode is characterized by a more complex movement, in which one can control the end effector of the robotic arm through the established kinematics, on three axes as bellow:

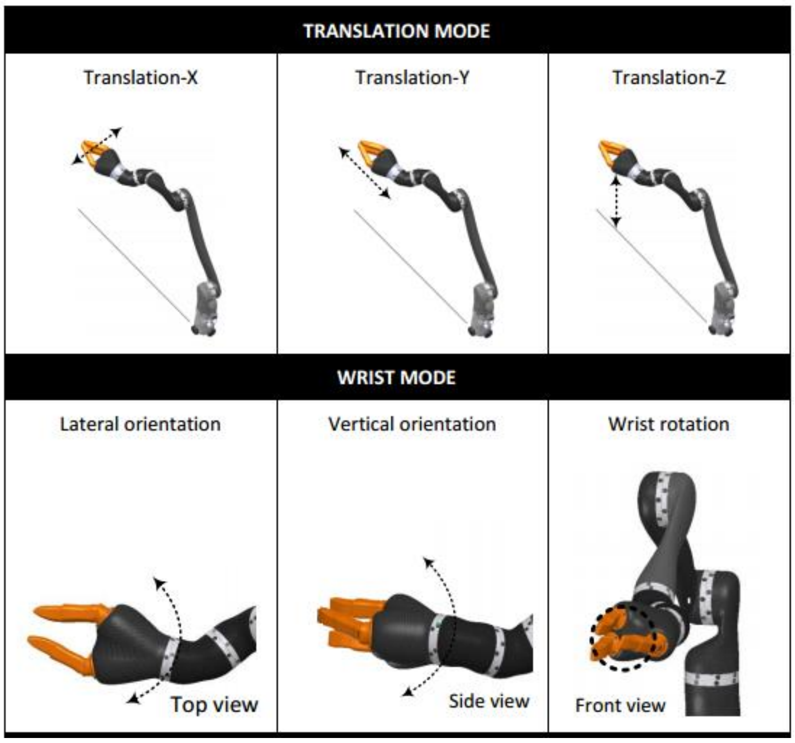


Figure 2.5 Cartesian Mode [2]

For this project, I used the angular mode, because the precision the movements was higher.

Also, Kinova's Intelligent Singularity Avoidance System always keeps Jaco² safely away from unwanted locations. JACO² is highly flexible and can adapt to all user needs. The detailed specifications are listed in the table below:

Table 2.4 Jaco Controller Specifications [2]

Controller Specifications	
Joystick	1 Mbps CANBUS
Power Supply	18 To 29 VDC
USB 2.0	12 Mbps
Ethernet	Not Available
Control System Frequency	100 Hz (High Level Api)
	500 Hz (Low Level API)
CPU	360 Mhz
SDK	
APIs	High and Low Level
Compatibility	Windows, Linux Ubuntu & ROS
Port	USB 2.0
Programming Languages	C++

2.1.4 *The Joystick*

The Kinova's standard controller is a 3-axis joystick mounted on a support which includes 5 independent push buttons and 4 auxiliary inputs (on the back side).

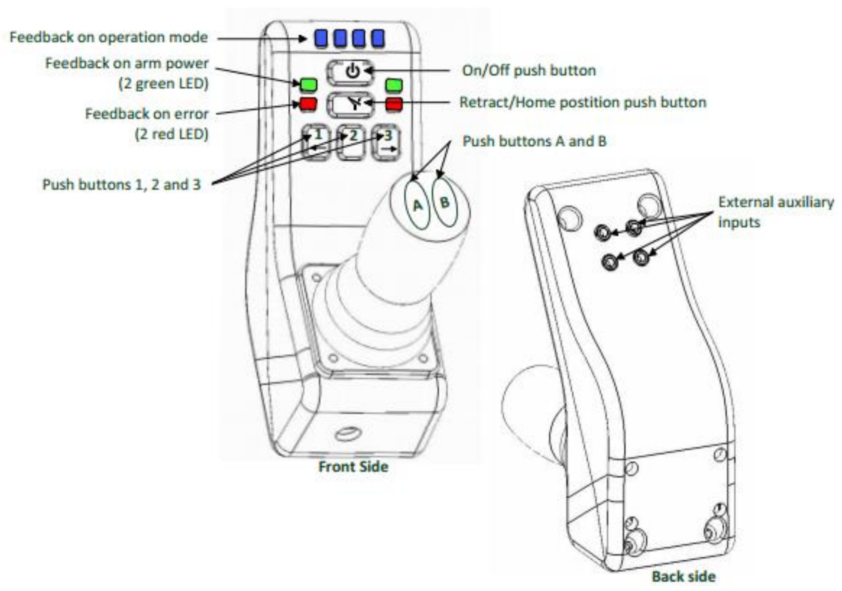
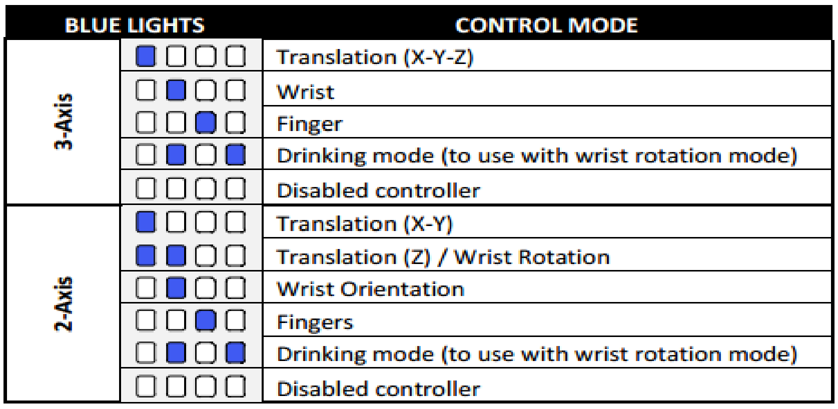

Feedback on operation mode
Feedback on arm power (2 green LED)
Feedback on error (2 red LED)
Push buttons 1, 2 and 3
On/Off push button
Retract/Home postition push button
Push buttons A and B
External auxiliary inputs
Front Side
Back side

Figure 2.6 Jaco – Joystick Specifications [2]

This Joystick allows the users to control the arm in two different operation modes Angular Mode and Cartesian Mode. In Cartesian mode, the arm may be manipulated using 2 or 3 axes.



BLUE LIGHTS		CONTROL MODE
3-Axis	🟦⬜⬜⬜	Translation (X-Y-Z)
	⬜🟦⬜⬜	Wrist
	⬜⬜🟦⬜	Finger
	⬜🟦⬜🟦	Drinking mode (to use with wrist rotation mode)
	⬜⬜⬜⬜	Disabled controller
2-Axis	🟦⬜⬜⬜	Translation (X-Y)
	🟦🟦⬜⬜	Translation (Z) / Wrist Rotation
	⬜🟦⬜⬜	Wrist Orientation
	⬜⬜🟦⬜	Fingers
	⬜🟦⬜🟦	Drinking mode (to use with wrist rotation mode)
	⬜⬜⬜⬜	Disabled controller

Figure 2.7 Jaco – Control Mode [2]

2.2 KINECT MODULE

The Kinect module represents a motion-sensing device, which was originally developed for the Xbox 360 gaming console. One of the distinguishing factors that make this device stand out among others in this genre, is that it is not a simple hand-controlled device, instead it can detect body position, motion and voice. Kinect provides a Natural User Interface (NUI) for interaction using body motion and gesture, as well as spoken commands. The controller that was once the heart of a gaming device, finds itself redundant in this Kinect age. For the latest releases of the Kinect, the controller is represented by the user itself.

This module has ushered a new revolution in the gaming world, and it has completely changed the perception of a gaming device. Since its inception, it has gone on to shatter several records in the gaming hardware domain. [12] It has now outgrown its Xbox roots and the Kinect sensor is no longer limited to only gaming.

Kinect for Windows is a specially designed PC-centric sensor that helps developers to write their own code, creating real-life applications with human gestures and body motions. With the launch of the PC-centric Kinect for Windows devices, interest in motion-sensing software development has scaled a new peak. Understanding the Kinect Device [12]. As Kinect blazed through the market in such a short span of time, it has also created a necessity of resources that help people learn the technology in an appropriate way. As Kinect is still a relatively new entry into the market, the resources for learning how to develop applications for this device are quite few. The Kinect module can be programed in C# and can interact with other devices using Kinect for Windows Software Development Kit (SDK).

Kinect is a horizontal device with depth sensors, color camera, and a set of microphones with everything secured inside a small, flat box. The flat box is attached to a small motor working as the base that enables the device to be tilted in a horizontal direction. The Kinect sensor includes the following key components:

• Color camera

• Infrared (IR) emitter

• IR depth sensor

• Tilt motor

• Microphone array

• LED

Apart from the previously mentioned components, the Kinect device also has a power adapter for external power supply and a USB adapter to connect with the computer. The following figure shows the above-mentioned components of the Kinect sensor:

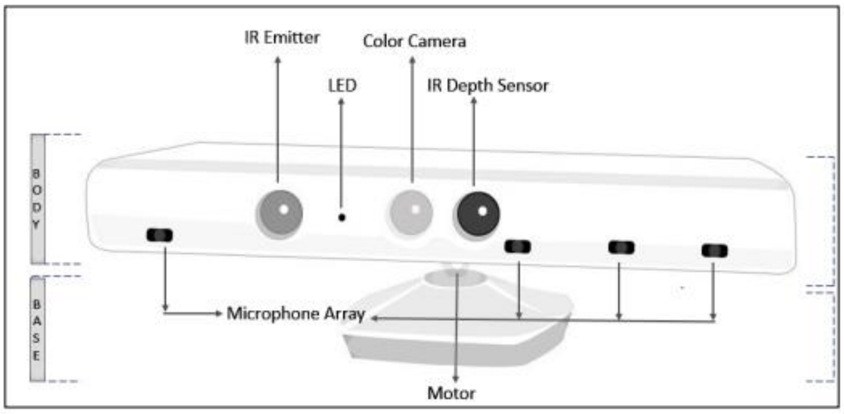


Figure 2.8 Kinect Architecture [12]

The following image shows how the Kinect looks in real-life, without the outer case:

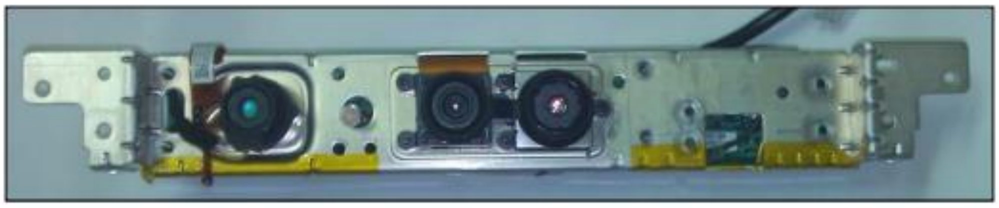


Figure 2.9 Kinect Architecture – reality

2.2.1 *The color camera*

The camera is used to capture and stream the color video data. Its function is to detect the primary colors form the source (RGB – Red, Green, Blue). It returns data stream consisting in a succession of image frames. The Kinect color stream supports a speed of 30 FPS at a resolution of 640 x 480 pixels. The value of FPS depends on the image resolution: the maximum resolution is 1280 x 960 at up to 12 FPS.

The viewable range for the Kinect cameras is 43 degrees vertical by 57 degrees horizontal, as shown below:

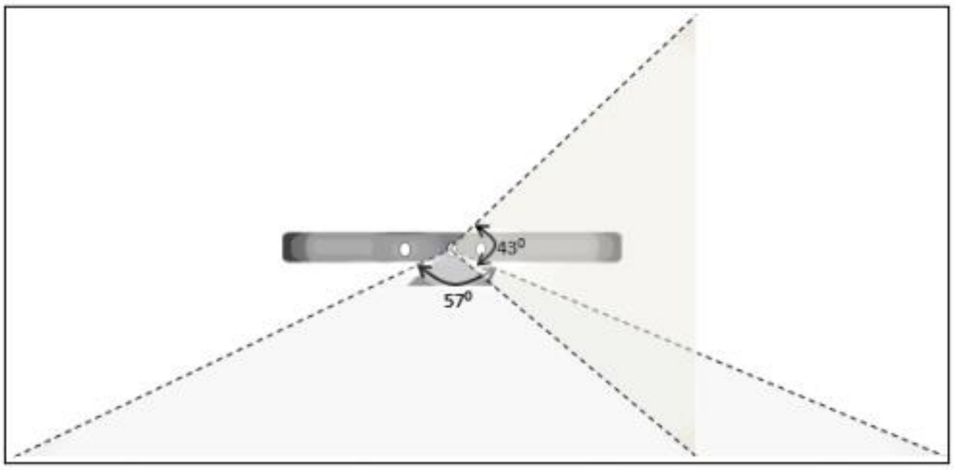


Figure 2.10 Sensitivity range of the Kinect camera [11]

2.2.2 *IR emitter and IR depth sensor*

The depth is represented by an IR emitter and an IR depth sensor working simultaneous. The Emitter is a projector of infrared light in a pseudo-random dot pattern over all the objects in front of it. These dots are invisible to humans, but they can provide depth information to the IR depth sensor. The dotted light reflects off different objects, and the IR depth sensor reads them from the objects and converts them into depth information by measuring the distance between the sensor and the object from where the IR dot was read [12]. The following figure shows how the overall depth sensing looks:

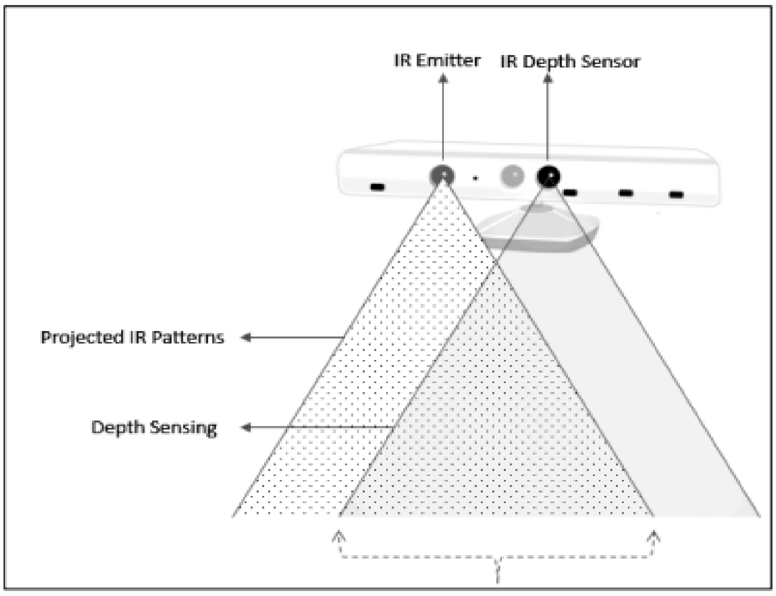


Figure 2.11 IR Emitter and Depth sensor [12]

Figure 2.12represents the image captured using this sensor:

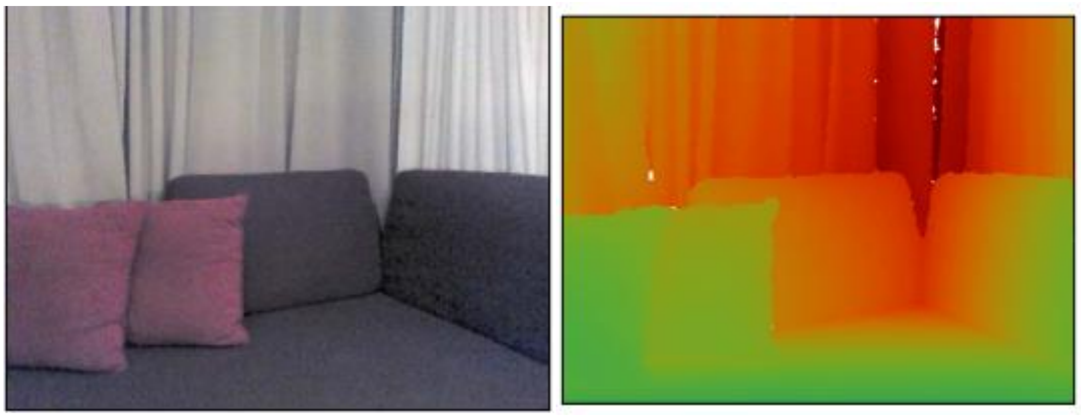


Figure 2.12 Depth image

2.2.3 Tilt Motor

The base and the body parts of the Kinect sensor are connected to a motor used to change the camera angles. The motor can be tilted vertically up to 27 degrees, so the Kinect sensor's angles can be shifted upwards or downwards by 27 degrees. The following figure shows an illustration of the angle being changed when the motor is tilted:

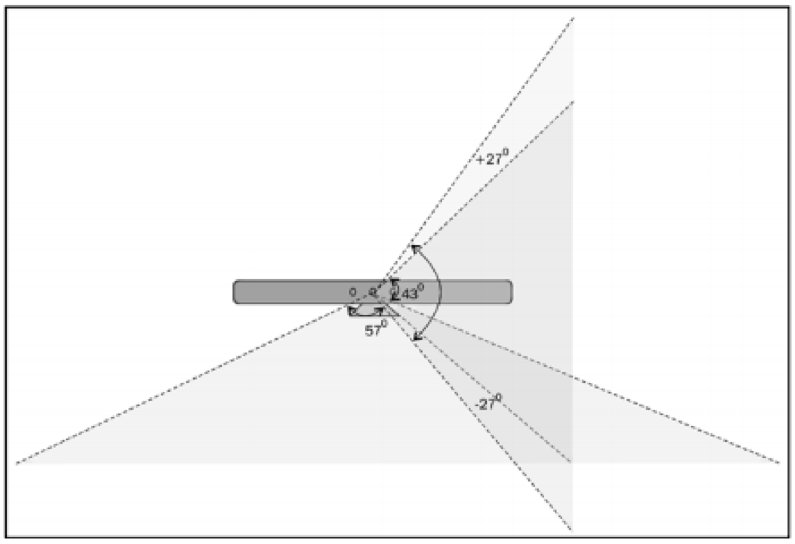


Figure 2.13 Titled motor [12]

2.2.4 Microphone array

The microphone array consists in four different microphones placed horizontally, three of them on the right side and one on the left side.

The purpose of the microphone array is not to just let the Kinect device capture the sound, but to also locate the direction of the audio wave. The main advantages of having an array of microphones over a single microphone are that capturing and recognizing the voice is done more effectively with enhanced noise suppression, echo cancellation, and beam-forming technology. This enables Kinect to be a highly bidirectional microphone that can identify the source of the sound and recognize the voice irrespective of the noise and echo present in the environment:

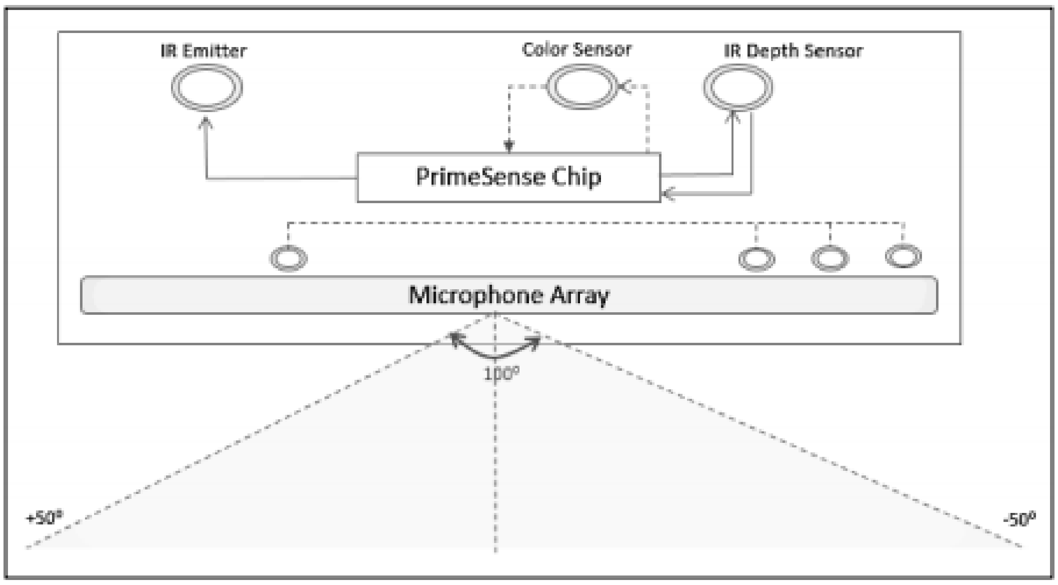


Figure 2.14 Microphone array [12]

2.2.5 LED

The LED is placed in between the camera and the IR projector. It is used for indicating the status of the Kinect device. The green color of the LED indicates that the Kinect device drivers have been loaded properly. If you are plugging Kinect into a computer, the LED will light up in green once your system detects the device.

2.3 HARDWARE

2.3.1 Arduino Uno

The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter.

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the GND and VIN pin headers of the Power connector. [3]

Table 2.5 Arduino Uno- Technical Specifications [3]

Technical Specifications	
Microcontroller	**ATmega328**
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB of which 0.5 KB used by bootloader
SRAM	2 KB
EEPROM	1 KB
Clock Speed	16 MHz

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 5V, the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

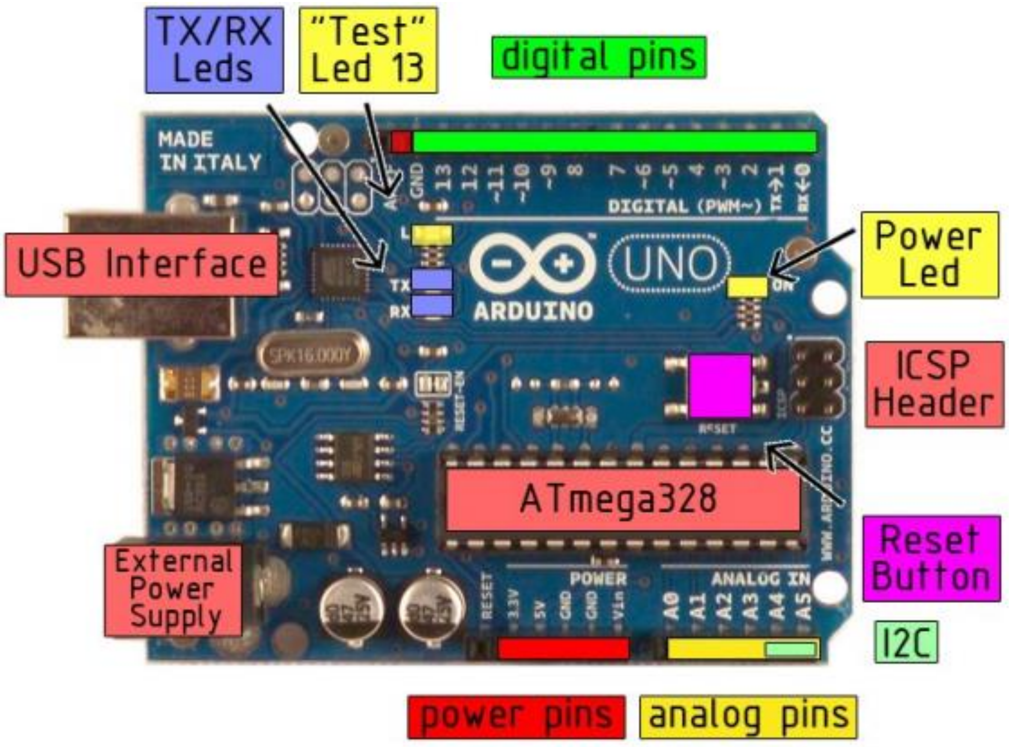


Figure 2.15 Arduino Uno architecture

The power pins are as follows:

- **VIN -** The input voltage to the Arduino board when it's using an external power source (different than 5 volts from the USB connection or other regulated power source)
- **5V -** The regulated power supply used to power the microcontroller and other components on the board.
- **3,3V** - A 3.3-volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND** - Ground pins.

All 14 digital pins on the board can be used as both input or output and operate at 5V. Each pin receives a maximum on 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhm. Some of the pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX) -** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- **External Interrupts: pins 2 and 3** - These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.
- **Pins 3, 5, 6, 9, 10, and 11** - 8-bit PWM.
- **SPI: Pins 10(SS), 11(MOSI), 12(MISO), 13(SCK) -** These pins support SPI interfacing.
- **LED Pin 13** - There is a built-in LED connected to the digital Pin 13. When the pin is set on HIGH value, the LED is on, when the pin is LOW, the LED is off. [3]

2.3.2 Servo motor DS04-NFC

The Servo DC gear motor is a 360-degree continuous rotation motor which can rotate continuously with both forward & backward. This motor has high torque and it is easy to interface it with any type of microcontroller. [11]

The Features of DS04-NFC Servo 360-Degree Continuous Rotation Servo DC Gear Motor are listed in the table below:

Table 2.6 DS04 – NFC Servo 360 - Technical Specifications [11]

Technical Specifications	
Model	**DS04-NFC**
Weight	38g
Dimensions	40.8 x 20 x 39.5 mm
Torque	5.5kg/cm (at 4.8 V)
Speed	0.22sec/60 °C (at 4.8V)
Operating voltage	4.8v-6v
Operating temperature	0 °C -60 °C
Current	1000mA

2.3.3 Hall Sensor A44 E

A Hall effect sensor is a transducer that varies its output voltage in response to a magnetic field. Hall effect sensors are used for proximity switching, positioning, speed detection, and current sensing applications. The stand of the Pyraminx has three magnets attached to it, 120 ° apart from each other. The motor stops when the magnet passes by the Hall sensor and the Kinect module takes a picture.

The Hall Module Sensor A44E can deliver an analogic or digital output, using LM393 comparator and the threshold voltage can be adjusted.

Table 2.7 Hall Module Sensor A44E - Technical Specifications [10]

Technical Specifications	
Supply Voltage	**4.5-24 [V]**
Output Saturation Voltage	400 mV
Output Leakage Current	10 µA
Supply Current	9.0 mA
Output Rise Time	2.0 µs
Output Fall Time	2.0 µs

CHAPTER 3 IMAGE PROCESSING

This step represents the data acquisition part of the project. The Kinect module captures some images that are later processed to find the color of all the pieces of the Pyraminx. This module returns a matrix of colors that serves as input for the solving algorithm.

The image processing block implies the following steps:

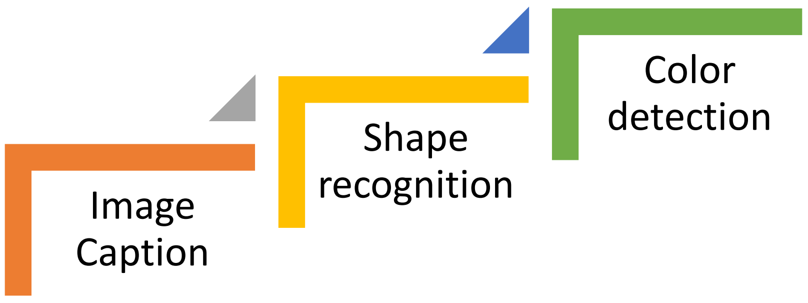


Figure 3.1 Data acquisition

3.1 IMAGE CAPTION

To be able to create a matrix that contains all the colors from the pyramid, it is not enough to have only one image of the puzzle. As my objective is to create an autonomous system I didn’t consider the option of moving the Kinect sensor manually to take pictures from different angles. The solution that I’ve implemented consists in rotating the stand that is holding the pyramid using a servo-motor and a Hall sensor connected to an Arduino board, as seen in Figure 3.2.

The system works as follows: the Kinect sensor takes a picture, the servo motor rotates the pyramid at 120°, then a second photo is taken. To be able to obtain all the colors from the puzzle, three photos are needed, one for every lateral face (the top face appears in all of them).

The electric scheme is shown below:

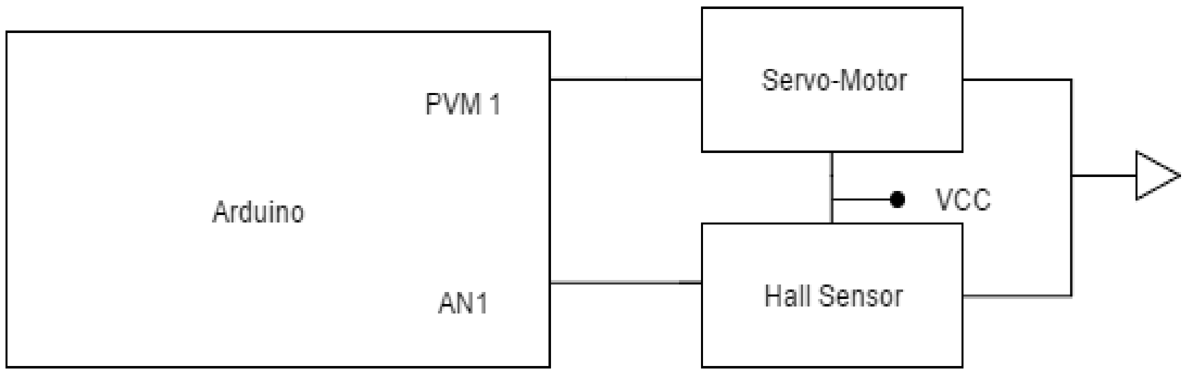


Figure 3.2 Servo-motor

3.2 SHAPE RECOGNITION AND COLOR DETECTION – METHOD I

Object detection and segmentation is the most important and challenging fundamental task of computer vision. It is a critical part in many applications such as image search, scene understanding etc. However, it is still an open problem due to the variety and complexity of object classes and backgrounds.

The images obtained at the previous step must be processed, to extract the color of every triangle. To do that, the first method I tried was a shape recognition algorithm that detects the vertexes of all the triangles from the image.

The shape recognition is implemented using OpenCV library and implies the following steps:

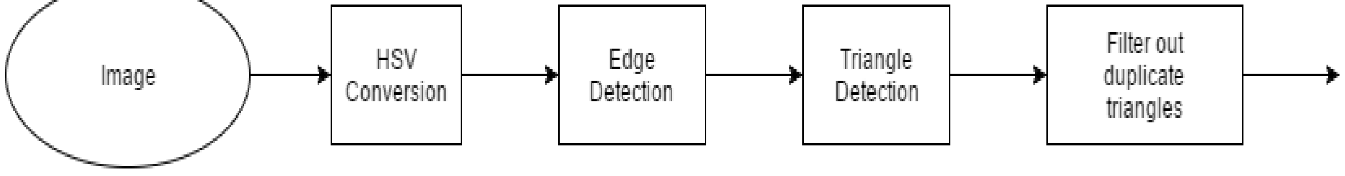


Figure 3.3 Shape recognition

3.2.1 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. This application was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code. [9]

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality etc.

For my project, I used this library to detect the edges of all the triangles on each face of the Pyraminx, as well as to recognize the colors of the pieces.

3.2.2 HSV conversion

HSV is one of the most used cylindrical-coordinate representation of points in a RGB model. This kind of representation implies a rearrangement of the geometry of the RGB to be more intuitive and perceptually relevant than the Cartesian representation. HSV, along with HSL were developed mainly for computer graphics applications, but nowadays are used in color pickers, image editing, image analysis and computer vision.

In HSV, the angle around the vertical axis represents the “hue” component, the distance from the axis corresponds to "saturation", and the distance along the axis corresponds "value" or "brightness".

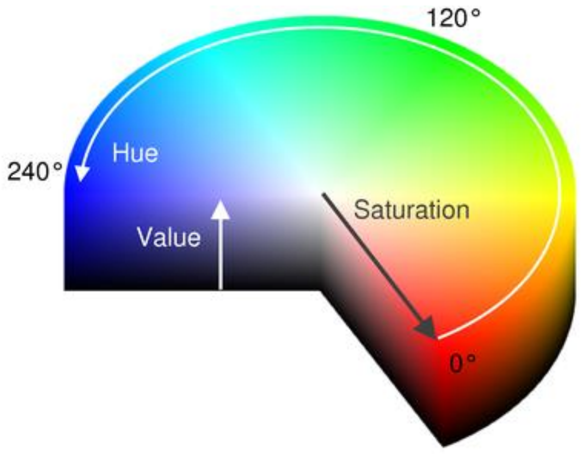


Figure 3.4 HSV Representation

The justification for choosing the HSV representation resides in the way the colors are distanced. Pyraminx colors are not corresponding to the ones obtained by the standard subtractive synthesis. The green is not exactly the basic, standard green, therefore using the RGB Representation, the means obtained for yellow and green will be very narrow and overlapping each other as values. The visual system of the Kinect is not as sensitive as it would be required to avoid misinterpreting of the pieces. After studying the options, I considered as a pertinent solution the HSV Representation. Since the coordinates are cylindrical instead of Cartesian, the confidence means will be broader and more reliable. For a better understanding, in what follows, **Table 3.1** had been drafted to illustrate the parallel between the RGB and HSV Representations.

Table 3.1 RGB Representation versus HSV Representation

	RGB Representation			**HSV Representation**		
Color quantity	Red	Green	Blue	Hue	Saturation	Value
Red Δ	130-255	0-162	0-155	0-9 / 151-180	1.00	1.00
Green Δ	0-50	150-255	0-50	46-100	0.875	0.795
Blue Δ	0-50	0-100	150-255	101-150	0.887	0.918
Yellow Δ	0-207	200-255	0-150	16-45	0.467	0.998

3.2.3 *Edge detection*

The purpose of this step is to find the contour of the Pyraminx in the image taken by the Kinect, so the task of triangle detection becomes easier. For this, I used Canny Edge Detection algorithm, which is a very precise technique to extract useful structural information from different vision objects. The algorithm implies several steps:

- Noise reduction – edge detection is very susceptible to noise, so the first step is to remove it with a 5x5 Gaussian filter.

- Finding Intensity Gradient of the Image (G) – The image is further filtered with a Sobel Kernel in both vertical and horizontal directions to obtain the first derivatives in each direction (G_x, G_y)

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = tan^{-1}\left(\frac{\mathrm{Gy}}{\mathrm{Gx}}\right)$$

- Non-maximum Suppression - After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge
- Hysteresis Thresholding – This is the decision step, which compares the gradient with a threshold value to determine the edges. [9]

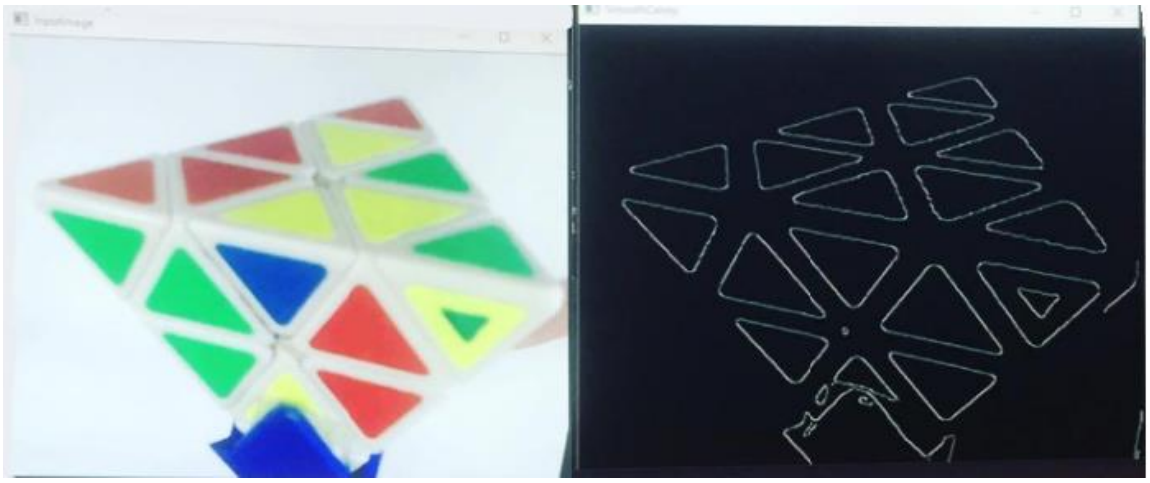


Figure 3.5 Canny Edge detection

3.2.4 *Triangle detection*

Using a shape approximation algorithm, I detect the vertexes of every triangle from the Pyraminx and compute the center of gravity for each of them. Around that point, I select a square of pixels on which I perform color detection, comparing the mean hue value of pixels with the intervals from **Table 3.1**. The triangle detection algorithm may return some vertexes twice, and the duplicate triangles must be filtered out. This is done by imposing the condition that all the triangles must be disjunctive.

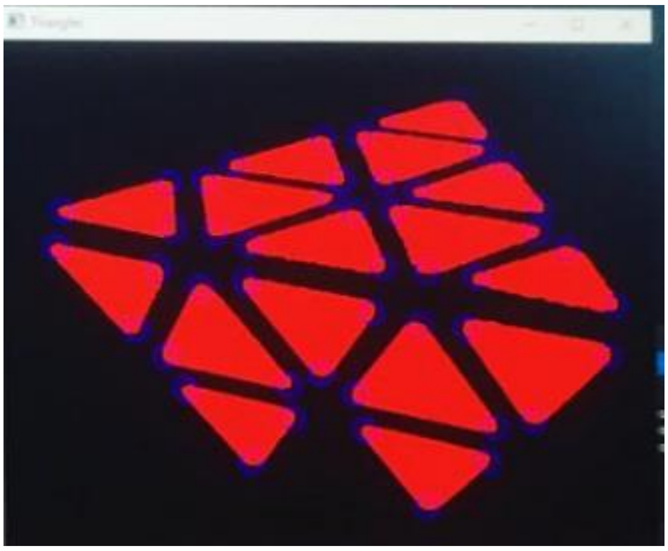


Figure 3.6 Triangle detection

3.2.5 Color detection

After the previous step is performed, color detection becomes a simple task: using the coordinates of the three vertexes of all the triangles, we can find the middle point of each of them.

We perform color detection on a small area around the middle of each triangle, using the HSV conversion. Finally, after all the processing the program will return a 5x12 color matrix.

Each row of the matrix represents a row of colors on the pyramid, and each 3 rows form a face of the puzzle. Here is an example for the top face. Each letter represents a color:

```
y 0 0 0 0

b y g 0 0

y r b b g
```

Using this method, I have obtained good results only in certain conditions: natural light and white background. Because the colors of the puzzle are not matte, if there is too much light, it will be reflected; in this case the recognition task becomes much more difficult and the error rate increases significantly.

From experimental point of view, I have observed that this algorithm does not distinguish between the yellow color of the triangle and the white contour of the Pyraminx, which lead to frequent and significant errors in the recognition task, even in optimal light conditions. Hence, I decided to try another method, presented the following sub-chapter.

3.3 SHAPE RECOGNITION AND COLOR DETECTION – METHOD II

To overcome the disadvantages from the previous method, I tried a totally different approach, based on machine learning techniques, starting from the observation that, depending on the light and angle, the colors of the pieces have very different properties, covering a wide range of shades. So, for each color I defined several classes, to cover all the possibilities. The idea is to train a system that can separate the colors form the Pyraminx, considering the colors to be discrete random variables. The separation is done based on two parameters of the image: the mean and the standard deviation. In this method, the whole image is analyzed. Afterwards, the color detection is performed and the last step consists in shape recognition.

3.3.1 Data Acquisition

To apply such an algorithm, I needed a data base consisting in different images with the Pyraminx, which I created using the Kinect Module. The data base contains 100 pictures of the puzzle, taken in different light setup (natural light and different neon light). Next, I have separated all the pieces from all the images based on their colors, using Paint.net, resulting in four images (one for each color) containing all the useful information from the data base. This images were used for training a system that discriminates colors. Figure 3.7 shows the training images for two colors.

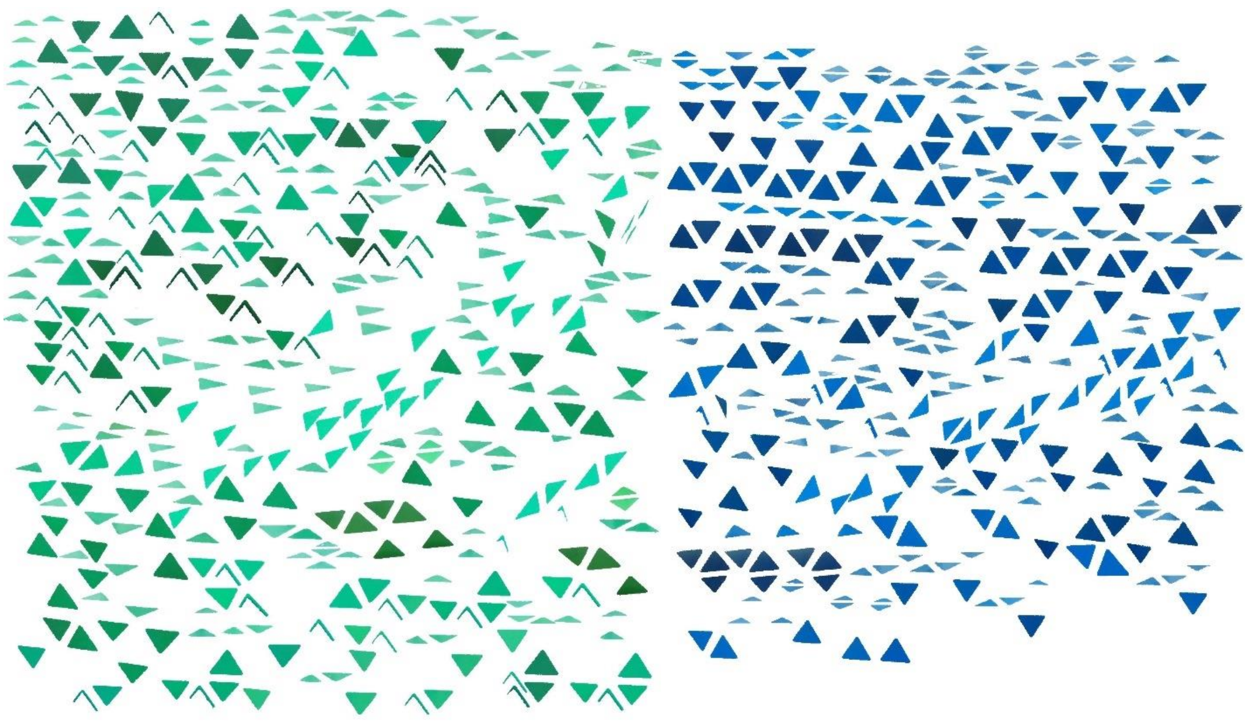


Figure 3.7 Train images- Green and Blue

3.3.2 Parameters

As mentioned above, the color discrimination is made based on two parameters:

- **Mean –** the mean value is computed for each component of the RGB: Red mean, Green mean and Blue mean for all the colored pixels from the training images and represents the expected value of that set. More practically, the expected value of a discrete random variable is the probability-weighted average of all possible values. In other words, each possible value the random variable can assume is multiplied by its probability of occurring, and the resulting products are summed to produce the expected value.
- **Standard Deviation (SD)-** is a measure that is used to quantify the amount of variation or dispersion of a set of data. A low SD indicates that the data points tend to be close to the mean of the set, while a high SD indicates that the data points are spread out over a wider range of values. The standard deviation of a random variable, statistical population, data set, or probability distribution is the square root of its variance. It is algebraically simpler, though in practice less robust, than the average absolute deviation. A useful property of the standard deviation is that, unlike the variance, it is expressed in the same units as the data. There are also other measures of deviation from the norm, including average absolute deviation, which provide different mathematical properties from standard deviation. [8] In this case, to minimize the errors I imposed a maximum value for SD.

3.3.3 K-means

As a second method used for color recognition I approached the K-Means clustering algorithm. This vector quantization process derived from the signal processing domain. In data mining the method is very popular for cluster analysis. Parsing n observations into K groups named "clusters" is made based on the proximity of the observation to the prototype of the array. This prototype is a matrix for the cluster and its characteristics become object of comparison. The obtained space that is populated with clusters is known as Voronoi space, and the clusters become Voronoi cells.

In computationally applications this method is very efficient. The algorithm needs initially random established centroids and from this point on the process itself is iterative and consists in moving the centroids. These centroids are chosen by color relevance. For the colors red, blue and green I used 3 centroids for each class, and for yellow, four. K-means moves the centroids to the average of the observations belonging to a cluster. By calculating the average of all the observations in the cell, the prototype will be moved to that specific position. This process repeats itself until no more moves are executed. The starting point can be inputted by the programmer or randomly chosen. for proper results, the K-means algorithm is recommended to be run multiple times but with different starting points to treat exhaustively each centroids path. The output is comparable based on the clusters distortion. The distortion is the sum of the squared differences between each observation and its allocated centroid. There is no perfect value for K but running it repeatedly with different values and other starting point, the generated result is examined and the clusters that don't make sense are determined. The K value also needs to be decreased if there are unpopulated cells.

Using this method, I obtained better results than with the previous one, but still there were problems regarding the recognition of the yellow color, because it reflected most of the light. Hence, I decided to replace the shiny stickers from the Pyraminx with matte ones and use a uni-color background when the Kinect module takes the pictures. This way, the color recognition works perfectly, regardless the light conditions. Having a one-color background, the shape detection task becomes trivial; I apply the triangle detection algorithm mentioned in section 3.2.4 and in this context, it works perfectly. The result is displayed in Figure 3.8:

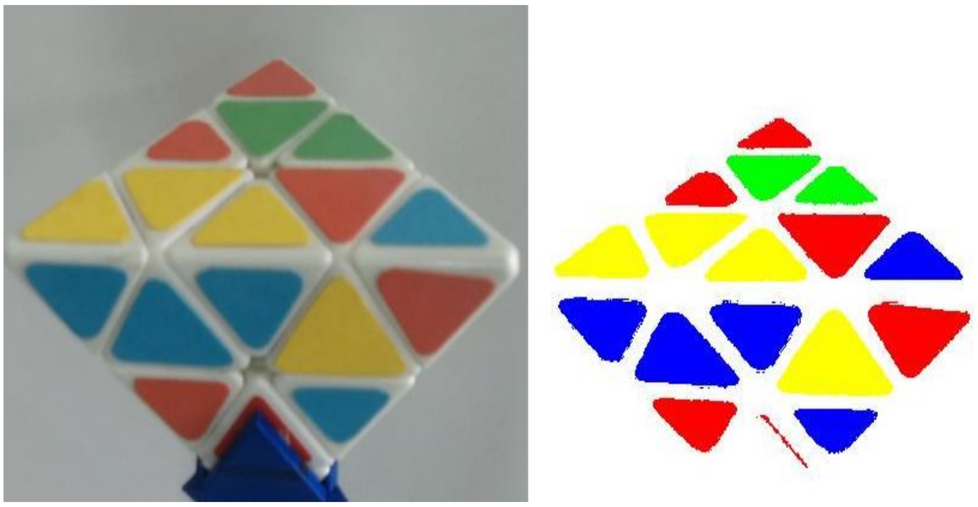


Figure 3.8 K-means Output

CHAPTER 4 SOLVING ALGORITHM

The Pyraminx is a complex puzzle in the shape of a regular tetrahedron, divided into 4 axial pieces, 6 edge pieces and 4 trivial tips. It can be twisted along its cuts to permute its pieces. The axial pieces are octahedral in shape, and can only rotate around the axis they are attached to. The 6 edge pieces can be freely permuted. The trivial tips are so called because they can be twisted independently of all the other pieces, making them trivial to place in solved position. [5]

The purpose of the Pyraminx is to scramble the colors, and then restore them to their original configuration.

4.1 SOLVING METHODS

There are many methods that can be used to solve the puzzle. Depending on what part one decides to solve first, the methods can be split into two main groups:

- V-first methods - two or three edges are solved first, and then some specific algorithms (LL algorithms) are applied to solve the rest of the pyramid
- Top-first methods - the top face, consisting in three edges around a corner, is solved first and the remaining is solved using a set of algorithms, keeping the top block in place.

Common V-first methods:

a) Layer by Layer - In this method, a face is solved first. This implies that a layer is solved as well. After that, the remaining puzzle is solved using algorithms particularly for this method.

b) L4E- L4E – is like Layer by Layer, the only difference is that two edges are solved around three centers.

c) Intuitive L4E – is the most advanced V-first method and requires a bigger amount of visualization capability. There are no specific algorithms, cubers try to intuitively solve each case by anticipating the movement of pieces.

Common Top-First methods:

a) One Flip - this method uses two edges around one center solved and the third edge flipped. There are six possible cases after this step, for which different algorithms must be executed. The third step involves using a common set of algorithms for all Top-First methods, called Keyhole last layer, which involves 5 algorithms, four of them being the mirrors of each other.

b) Keyhole- it uses two edges in the right place around one center and the third edge does not match any color of the edge (i.e. it is not in the right place or is flipped). The centers of the fourth color are then solved. The last step is solved using Keyhole last layer algorithms.

c) OKA- one edge is oriented around two edges in the wrong place, but one of the edges that is in the wrong place belongs to the block itself. The last edge is found on the bottom layer and a very simple algorithm is executed to get it in the right place, followed by keyhole last layer algorithms.

All the methods described above have some advantages and disadvantages, depending on how the pieces are distributed. Usually, professional cubers learn all the methods and while observing a case, they decide which method best suits that case. [4]

4.2 PYRAMINX MOVES

For this thesis, I chose to implement my own method of solving the Pyraminx, which implies four main steps that will be further explained in this paper:

- Step 1: get all centers in the correct position
- Step 2: get all corners in the correct position
- Step 3: solve the top face
- Step 4: solve the rest of the pyramid

In order to permute all the pieces, a minimum of four moves that will be performed by the robotic arm is required. Using only these moves in different combinations, one can solve the Pyraminx in all situations. I defined the moves in only one direction, (clockwise for move 1 and 2 and counter-clockwise for move 3); the reverse move is equivalent with two successive moves. For the simplicity of the algorithms, I assigned a number to every face of the pyramid and to every move, as shown in the figure below.

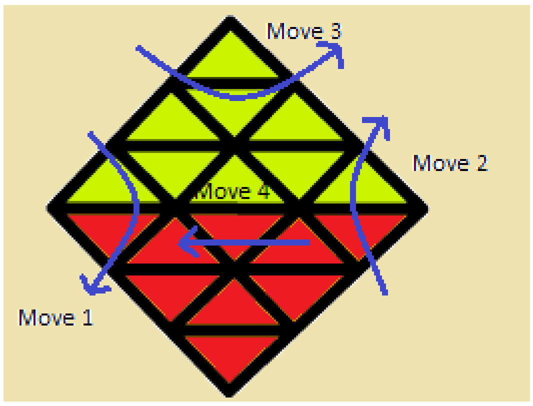


Figure 4.1 Pyramid Faces and moves

- *Move 1*: means that that the 4th face (green one), will go over the first one (yellow), the second will go over the fourth, and the first over the second, like so:

  $4 \rightarrow 1$

  $2 \rightarrow 4$

  $1 \rightarrow 2$

- *Move 2*: following the same principle as above:

  $2 \rightarrow 1$

  $1 \rightarrow 3$

  $3 \rightarrow 2$

- *Move 3*: this time the move is counter clockwise:

  $4 \rightarrow 1$

  $3 \rightarrow 4$

  $1 \rightarrow 3$

- *Move 4:* represents the rotation of the first layer:

  $2 \rightarrow 4$

  $4 \rightarrow 3$

  $3 \rightarrow 1$

The first three moves represent permutations of the edges or corners, and the forth move means the clockwise rotation of the top layer. To implement the algorithm, one must know how each move affects the position of the pieces, in order to compute the necessary moves required to solve the Pyraminx. To have complete information about every piece of the puzzle, I stored the information regarding the position of every tringle in a three-dimensional vector, called *fete [faceNumber][LineNumber][ColumnNumber]*, where the *lineNumber* and *columnNumber* represent the position of the piece and face number is pointing to a certain face as you can see in the figure below.

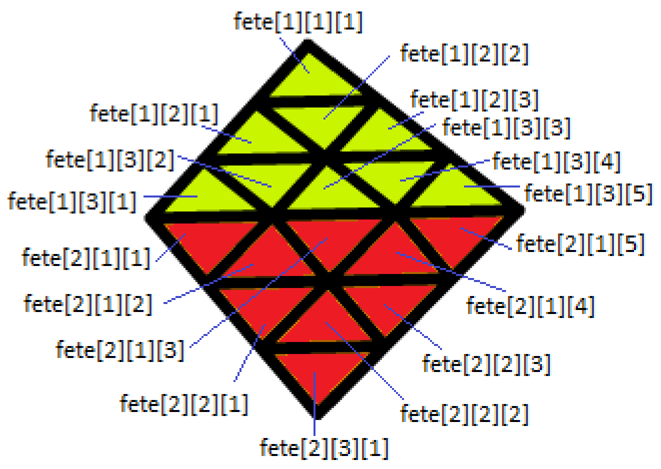


Figure 4.2 Position of the pieces

The yellow face will be positioned on top, as in the image above, and the counting will start from the top corner. For the remaining faces, the counting will start in the reverse way, beginning with the biggest edge, as they would naturally be looked at.

4.3 Solving algorithm - centers and corners

The first step in solving the Pyraminx is to know, even if the pieces are scrambled, what color corresponds to each face. There are two easy ways to figure out what is the color of one face: if one can match three center pieces having the same color on one face, that is the color of the face. The other solution is concentrating on the corner pieces. All three corner pieces from a face can be positioned to match color, and that is the color of the face. [6]

For my project, the pyramid will always be placed in the stand with the yellow face on top, so the face recognition step will be skipped. The second step is to correctly position the center pieces, and that is done using a simple function. The verification must be done over just one face. If the centers are correctly positioned on one face, all the other center pieces will be in place. The same principle is applying also for corners.

After all the centers are in the right position, the task of matching the corners becomes very simple: the corner color must match the one from the associated center. After this step is complete, the pyramid should look like in **Figure 4.3**.

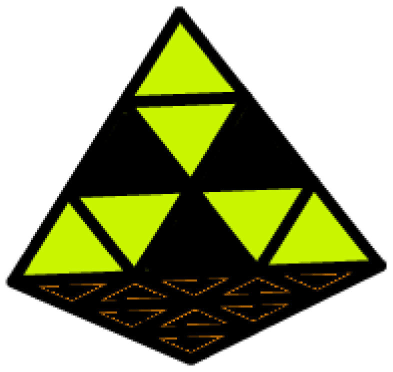


Figure 4.3 Centers and Corners

4.4 SOLVING ALGORITHM – TOP FACE

4.4.1 Get Piece on the second Layer

After the centers and the corners are in place, the next step is to put the rest of the yellow edge pieces in their position. The algorithm for that is more complex and multiple possibilities must be taken into consideration. There are 12 possible positions for a side edge piece, and only one of them is the correct one. [7]

From this point on, since only side-edge pieces must be permuted, I redefined the moves presented in section 4.2 moves only for the side-edge pieces, and will be called *sides*. **Figure 4.4** shows how the sides are numbered. The first number represent the face and the second the position. Each face has three sides.

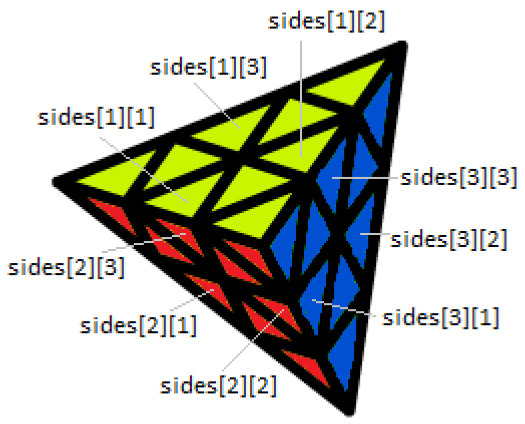


Figure 4.4 Sides

Theoretically, there are 12 different sides, but practically, two by two represent the same piece. The connections between the sides are the following:

- side [1][1] = side [2][3];
- side [1][2] = side [3][3];
- side [1][3] = side [4][2];

- side [2][1] = side [4][1];
- side [2][2] = side [3][1];
- side [3][2] = side [4][3].

Using the sides, and considering the actual pieces, the moves can be redefined as:

- *Move 1:*

  side [1][3] $\longrightarrow$ side [4][1]
  side [1][1] $\longrightarrow$ side [4][2]
  side [4][1] $\longrightarrow$ side [2][3]
  side [4][2] $\longrightarrow$ side [2][1]
  side [2][3] $\longrightarrow$ side [1][3]
  side [2][1] $\longrightarrow$ side [1][1]

- *Move 2:*

  side [1][2] $\longrightarrow$ side [2][3]
  side [1][1] $\longrightarrow$ side [2][2]
  side [2][3] $\longrightarrow$ side [3][1]
  side [2][2] $\longrightarrow$ side [3][3]
  side [3][1] $\longrightarrow$ side [1][2]
  side [3][3] $\longrightarrow$ side [1][1]

- *Move 3:*

  side [1][3] $\longrightarrow$ side [4][3]
  side [1][2] $\longrightarrow$ side [4][2]
  side [4][3] $\longrightarrow$ side [3][3]
  side [4][2] $\longrightarrow$ side [3][2]
  side [3][3] $\longrightarrow$ side [1][3]
  side [3][2] $\longrightarrow$ side [1][2]

- *Move 4:*

  side [1][1] $\longrightarrow$ side [1][2]
  side [1][2] $\longrightarrow$ side [1][3]
  side [1][3] $\longrightarrow$ side [1][1]
  side [3][3] $\longrightarrow$ side [4][2]
  side [4][2] $\longrightarrow$ side [2][3]
  side [2][3] $\longrightarrow$ side [3][3]

The first possibility that I am considering in this algorithm is the one in which the desired side is placed on the top face, but not in the right position. In this case, the piece must be lowered on the second layer, and later positioned correctly on the first layer. Depending on its position, a different combination of moves is required to lower a certain piece. All the possible cases are illustrated below, together with the set of moves required to move that piece, keeping the centers and the corners in place. A number followed by the sign “ ‘ ”, means the counter direction move, or twice the same move.

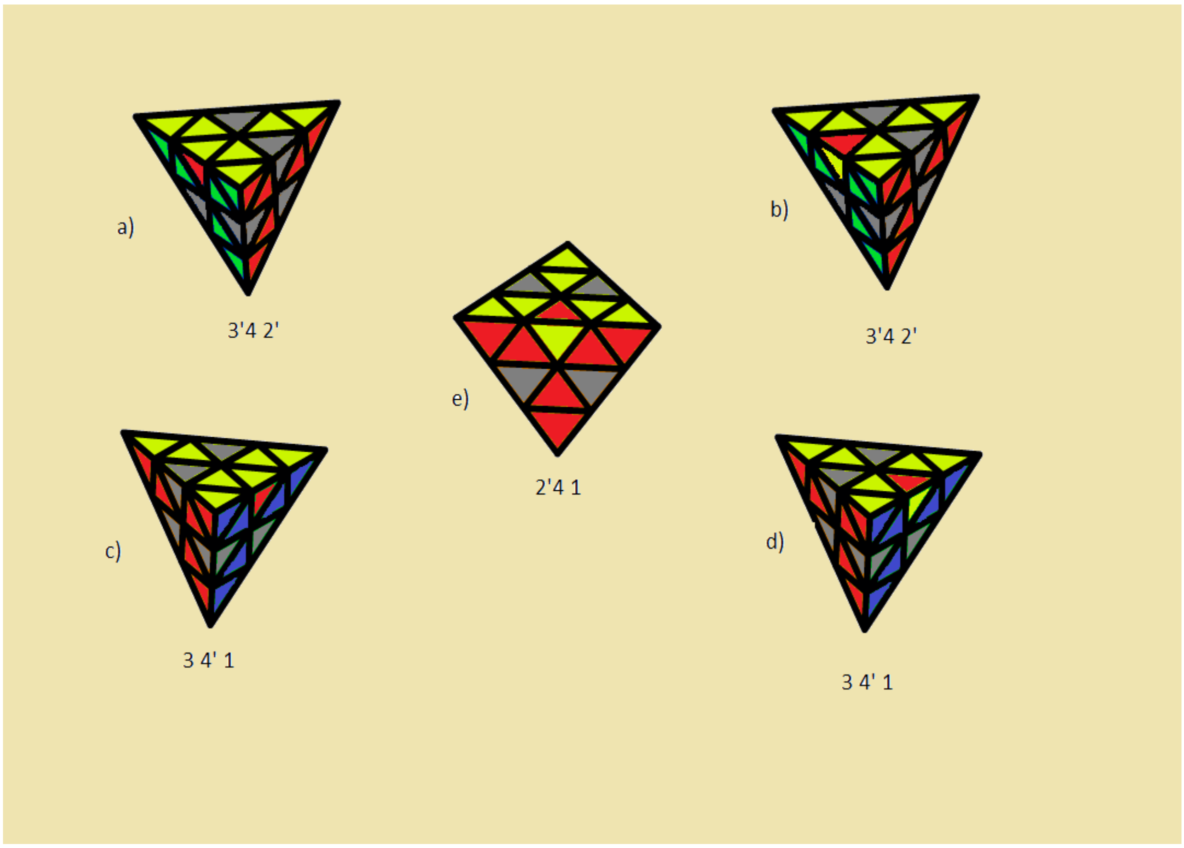


Figure 4.5 Get Piece on Layer 2 – side [1][1]

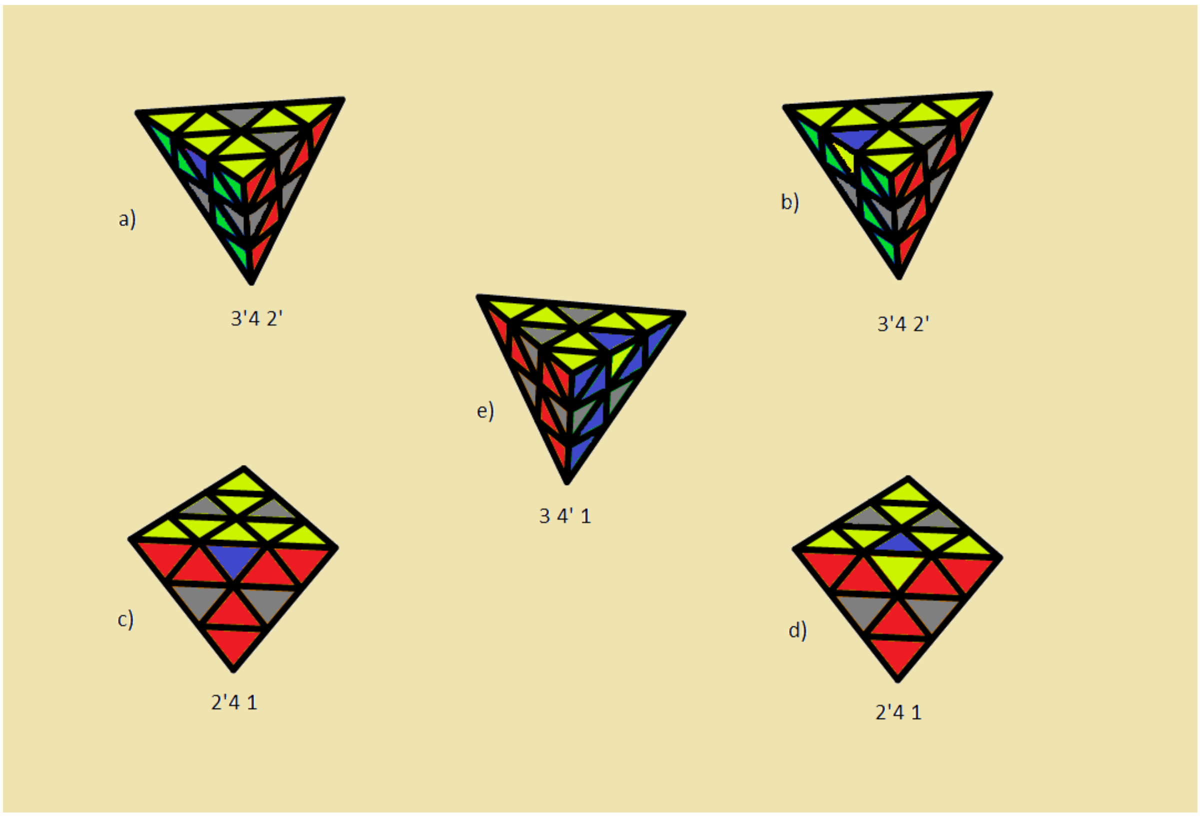


Figure 4.6 Get Piece on Layer 2 – side [1][2]

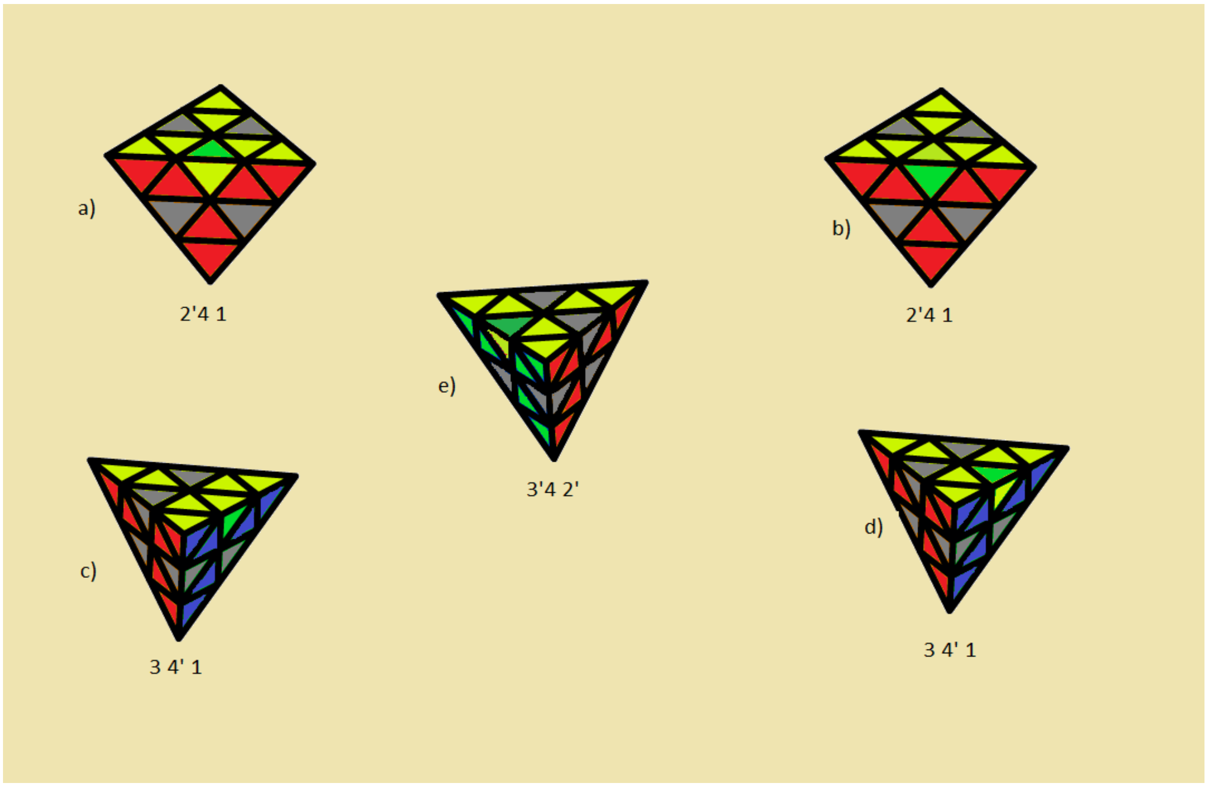


Figure 4.7 Get Piece on Layer 2 – side [1][3]

4.4.2 *Solving the top face*

After all the sides are placed somewhere on the second layer, one can focus on solving the top face. The goal is to position all the yellow sides in the right place, without changing the centers and the corners, and of course without changing the position of the other sides from the top face (otherwise this step will become an infinite loop). Because I have already eliminated five possible positions (on the top layer), only six cases are left for each side.

For this step, it is useful to define some fundamental sets of moves, that will be used to move a certain side into the right position. This moves, relative to Face 2 are described in the figure below:

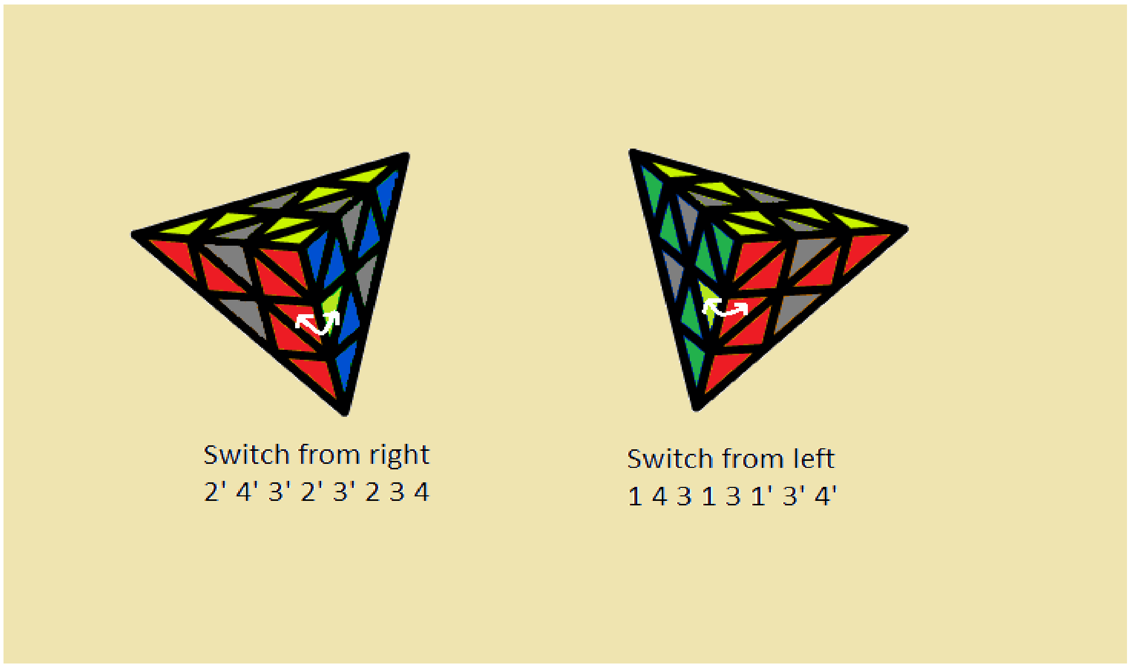


Figure 4.8 Fundamental sets 1 and 2

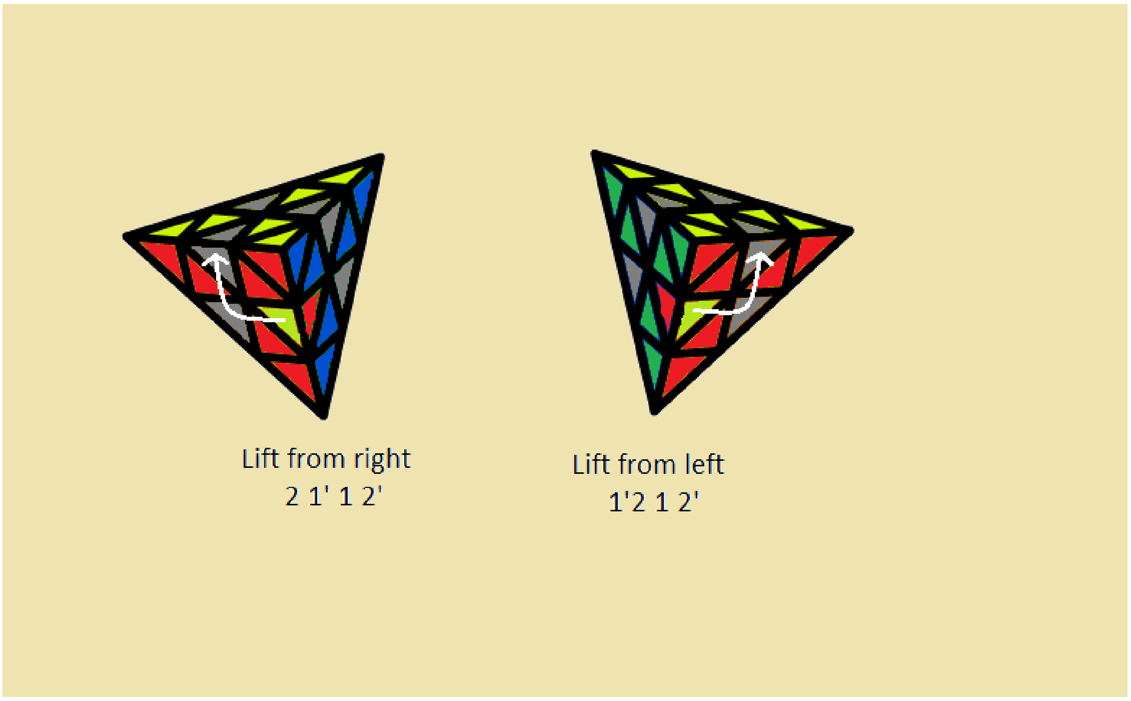


Figure 4.9 Fundamental sets 3 and 4

With this fundamental sets, one can lift the yellow and red side from Face 2, to its correct position on the top face. Applying the same principle, I was able to translate these sets in relation with the other two faces as follows:

- *Translation relative to Face 3:*

  $1 \longrightarrow 2$

  $4 \longrightarrow 4$

  $3 \longrightarrow 1$

  $2 \longrightarrow 3'$

- *Translation relative to Face 4:*

  $1 \longrightarrow 3'$

  $2 \longrightarrow 1$

  $3 \longrightarrow 2'$

  $4 \longrightarrow 4$

Using the above fundamental moves and the respective translations, Face 1 can be solved entirely. This step normally consists in six cases per side, but I also included here the case in which the piece is reversed on its position, because it's simpler to directly reverse the piece using a fundamental move set, than to firstly lower it on the second layer and then put it back in the correct position. This gives a total of seven cases per side. The moves combination is listed below for all the sides.

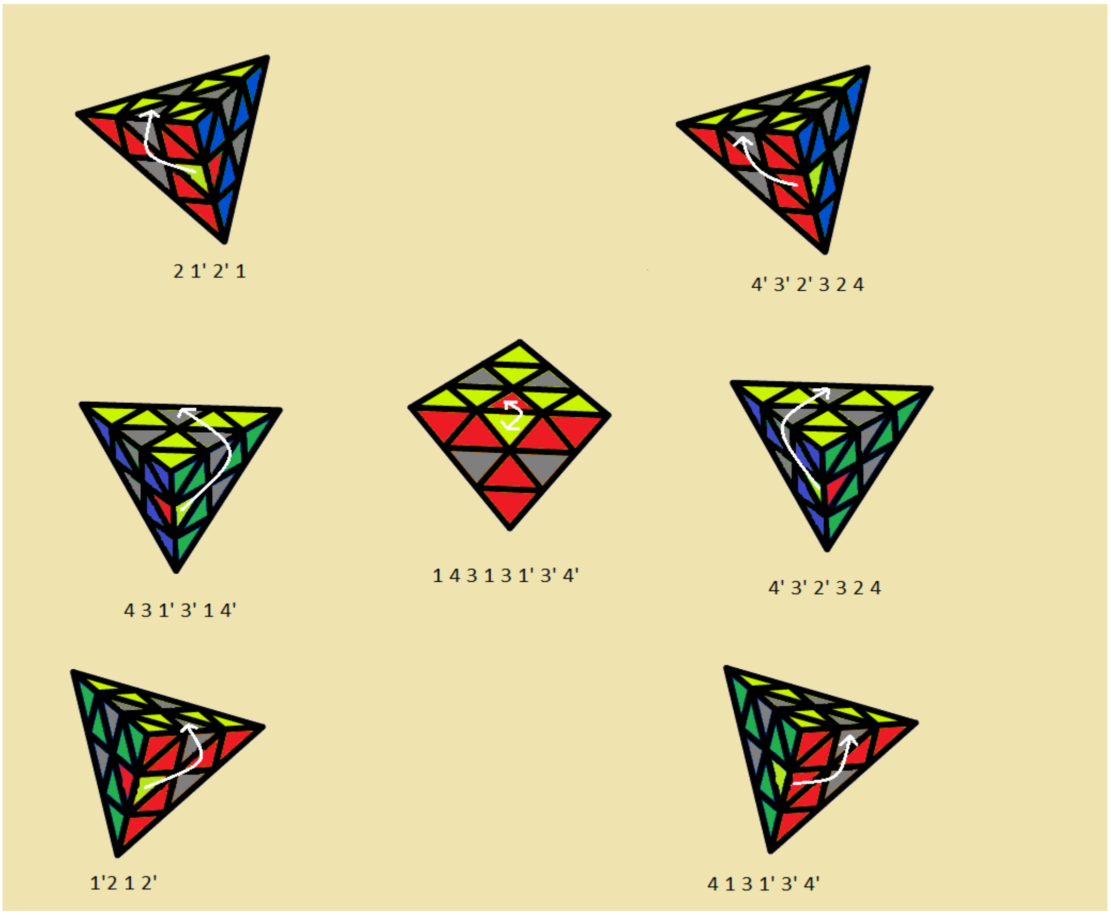


Figure 4.10 Get side [1][1] in place

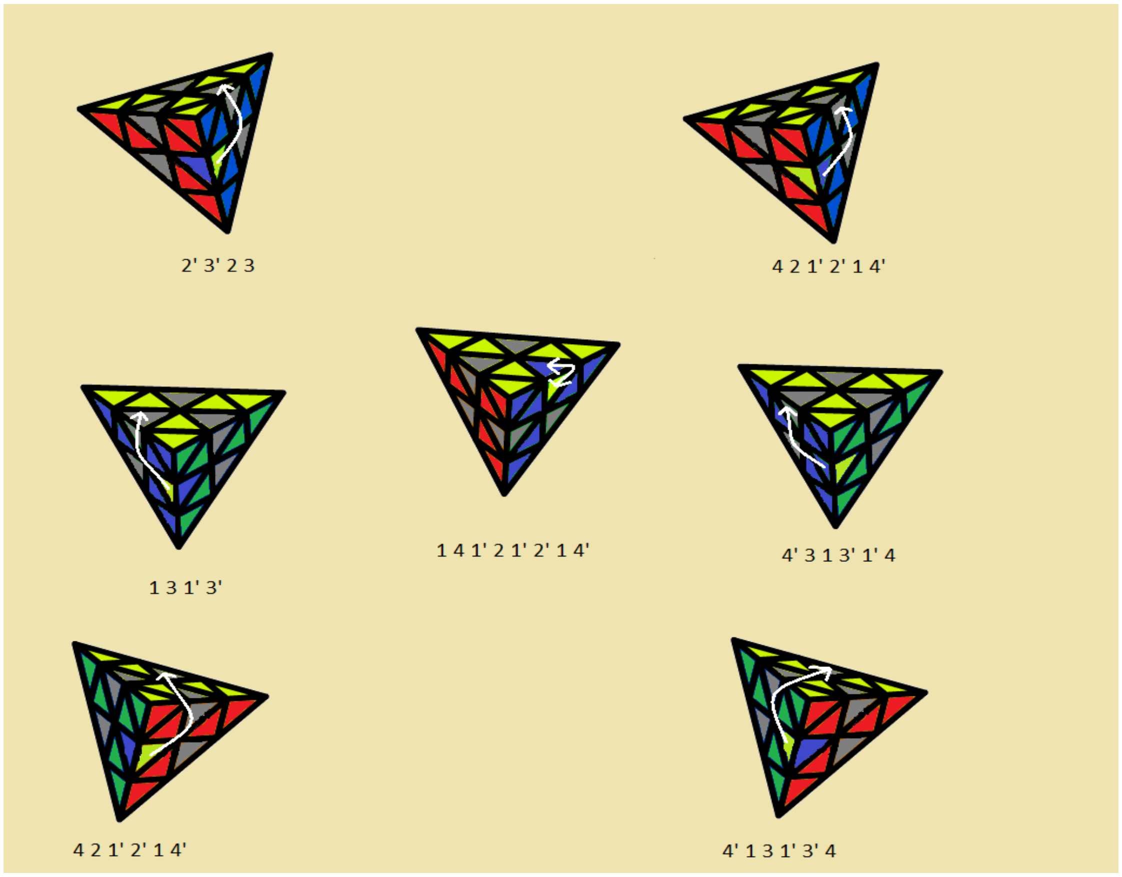


Figure 4.11 Get side [1][2] in place

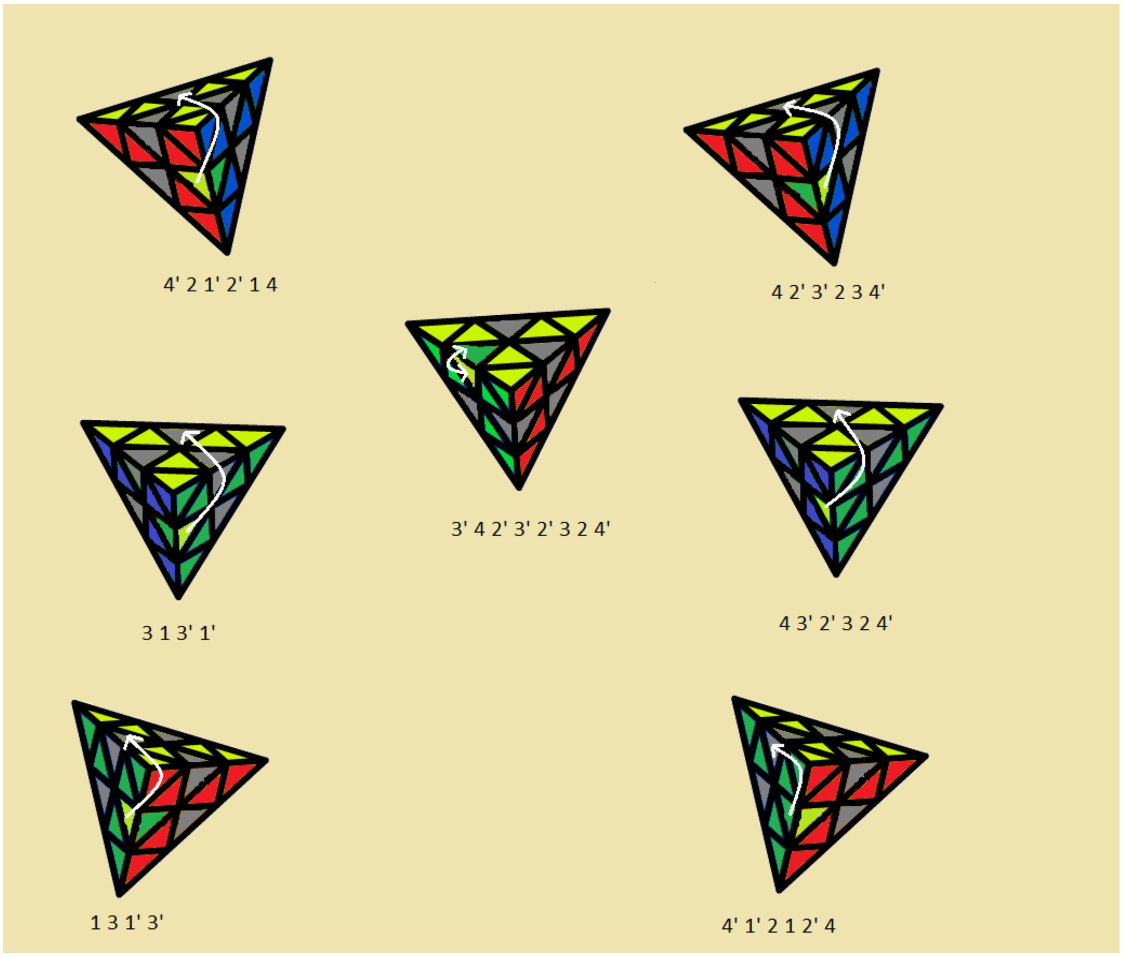


Figure 4.12 Get side [1][3] in place

4.5 SOLVING ALGORITHM – PLACING THE LOWER EDGES

After the top piece is solved, there are only three lower edges that can be out of place. Depending on their position, four different cases can occur:

- *Case A*: In the most fortunate case, after solving the top face, all the lower edge pieces are in the right place and the pyramid is solved.

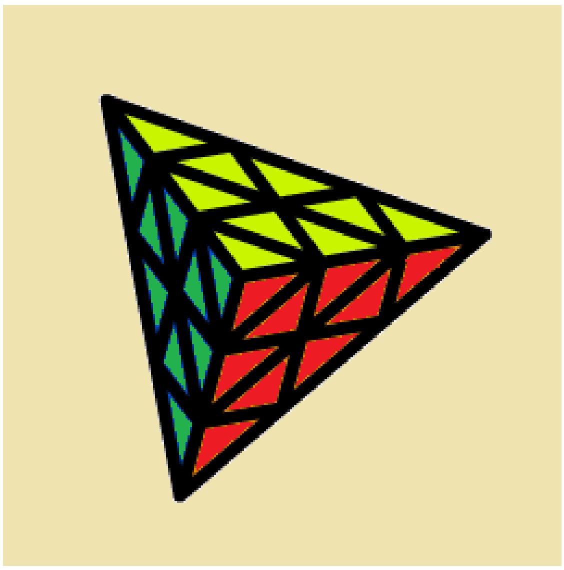


Figure 4.13 Case A

- *Case B*: one edge piece is in the correct position and the other two need to be flipped. There are three possible scenarios:

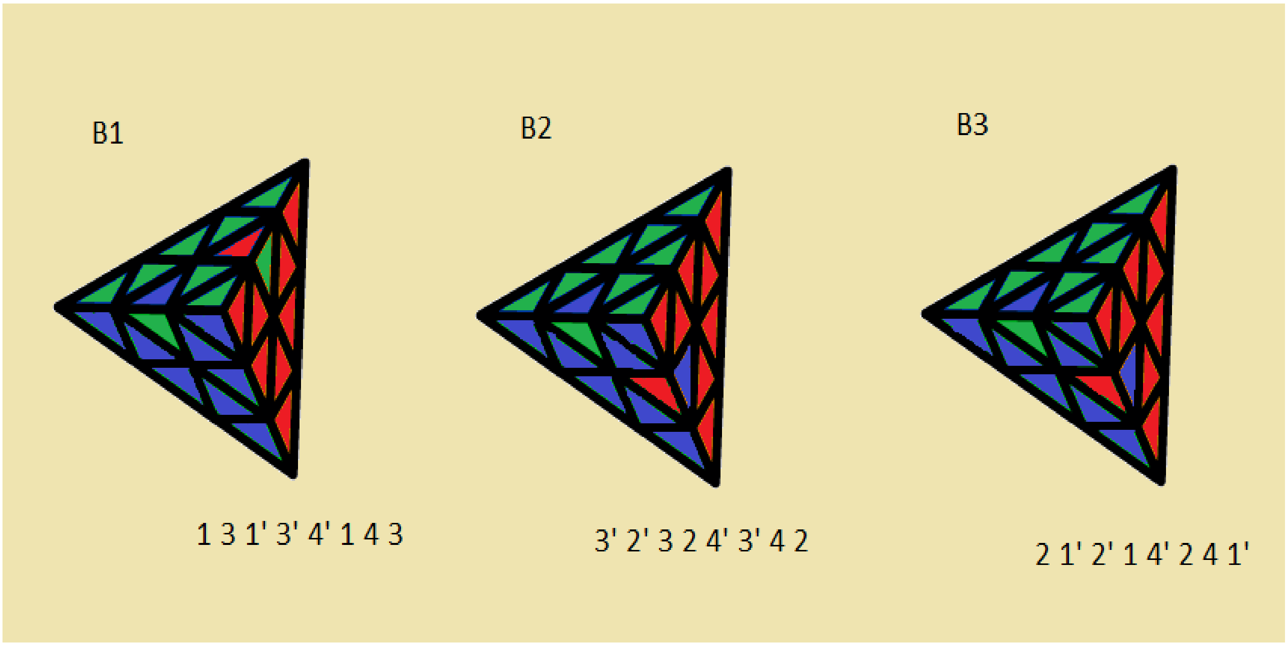


Figure 4.14 Case B

- *Case C*: one edge piece is wrong and the other two have one color that lines up with the adjoining side. To determine this case, it is enough to impose some simple conditions as follows:

  side [2][2] ≠ red and side [3][1] ≠ blue – case B1;

  side [2][1] ≠ red and side [4][1] ≠ green – case B3;

  side [3][2] ≠ blue and side [4][3] ≠ green – case B2.

  In this case, one must apply the same algorithm as in case B, but the result won't be the solved pyramid, instead it will look like a Case D.

- *Case D*: All three edge pieces are completely wrong but when you turn the top corner, they will line up with the bottom perfectly. There are two possibilities in this scenario, as shown in the figure below, depending on the direction, the edges must be changed.

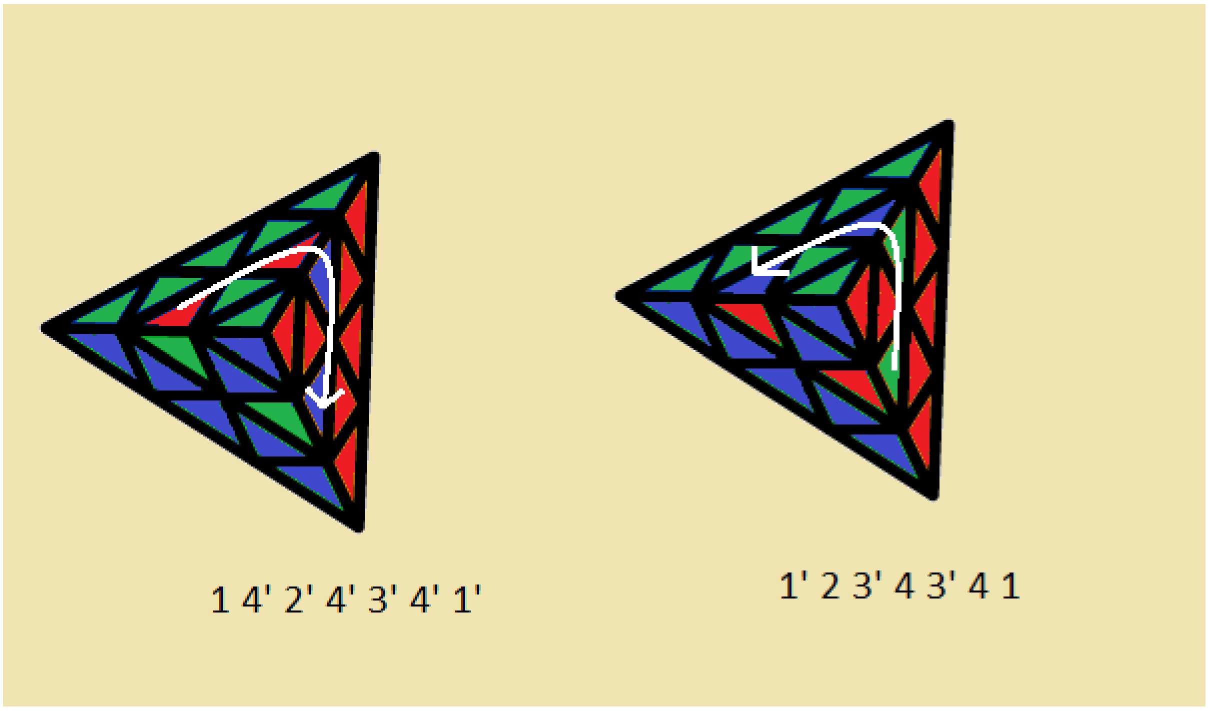


Figure 4.15 Case D

Using the algorithms described above, the three remaining edges will be placed in the correct position, without altering the top face. The pyramid is now solved.

CHAPTER 5 PERFORMING THE MOVES

5.1 GENERAL DESCRIPTION

This part represents the interface with the robotic arm, transposing the set of movements computed by the algorithm into pure movement data, passed to the Kinova's controller. The input will consist in a string of numbers, each number being corelated with one of the four moves.

Since all the moves are complex and require high precision, the position of all the actuators are updated during a move, so the action is smooth and as natural as possible. The actual programming of the robotic arm requires two steps:

- Manipulate the arm using the joystick in angular mode as specified in Chapter 2. I firstly performed the movement controlling the arm with the joystick and record the position (angles) of all the actuators in different points.
- Program the hand to automatically perform the action, using the `PointToSend` function (from Kinova.dll), which receives as parameter an angle (how many degrees the actuator rotates from its previous position). For the movement to look natural and coherent, the actuators are updated at different moments of time.

5.2 MOVEMENT IMPLEMENTATION

To completely solve the Pyraminx, five complex moves are required: *Rotate_Layer*, *Move_BigCorner1*, *Move_BigCorner2*, *Move_BigCorner3*, *Move_SmallCorner*. The first for moves correspond with the moves defined in Chapter 4 as follows:

- *Move_BigCorner1* ⟶ *Move 1*
- *Move_BigCorner2* ⟶ *Move 2*
- *Move_BigCorner2* ⟶ *Move 3*
- *Rotate_Layer* ⟶ *Move 4*
- *Move_SmallCorner* ⟶ *Represents the rotation of the corner facing the robotic arm.*

5.2.1 Home position

For each move, the arm starts from a default initial position called home position, having the following parameters:

Table 5.1 Home position parameters

Home position parameters	
Actuators	**Degrees (°)**
Actuator 1	275.29
Actuator 2	167.39
Actuator 3	57.13
Actuator 4	241.02
Actuator 5	82.7
Actuator 6	75.75
Finger 1	0
Finger 2	0
Finger 3	0

The picture below shows the initial position of the arm:

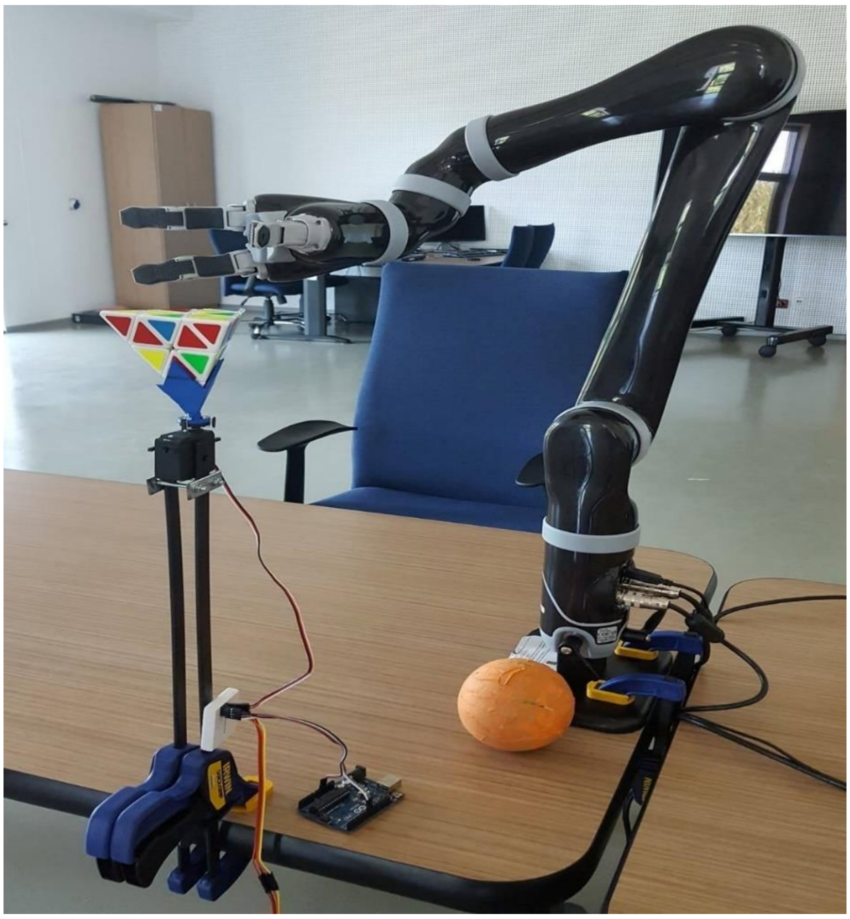


Figure 5.1 Home position

5.2.2 *Move_BigCorner1*

Starting from Home position, this move rotates the big corner placed on the right side of the robotic arm. This is the most difficult move I've implemented, because of the relative position between the arm and the puzzle. The final position of the actuators was experimentally found, using the joystick and the monitor function of Kinova Development Center API. This values represent the final position of the robot for this move and their values are listed in the table below.

By subtracting this new coordinates of each actuator from the values from home position, I deduced how many degrees each actuator must rotate, which is the parameter I need in my code.

Table 5.2 Move_BigCorner1

Actuators	Final position (°)	Relative difference (°)
Actuator 1	234.49	-40.8
Actuator 2	285.09	117.7
Actuator 3	233.6	176.47
Actuator 4	122.93	-188.09
Actuator 5	181.09	98.39
Actuator 6	-390.55 -> -30.55	-338.45 -> 21.55
Finger 1	4776	-
Finger 2	4788	-
Finger 3	4800	-

The fingers positions are updated using only a number between [0;6000], 0 meaning the fingers are fully opened, and 6000 meaning they are fully closed. After the robot reaches this position, changing the value of the Actuator 6 with -121.34 ° (using the same principle as above: find with the joystick the new final position of Actuator 6 and subtract from it the previous value), the robot performs Move 1 from Chapter 4. After the action is completed, the hand returns to Home position.

Since Actuator 6 has a range of [-1000 ° - +1000 °], it can rotate more than a full circle. So, the value -390.55 ° can be reduced to the first quadrant: -390.55° + 360 ° = -33.55 °. Obviously, the sign of the angle dictates the direction of the rotation. I chose the order in which the actuators move so that the arm does not interact with the stand or the Kinect module. The image below shows the position of the arm:

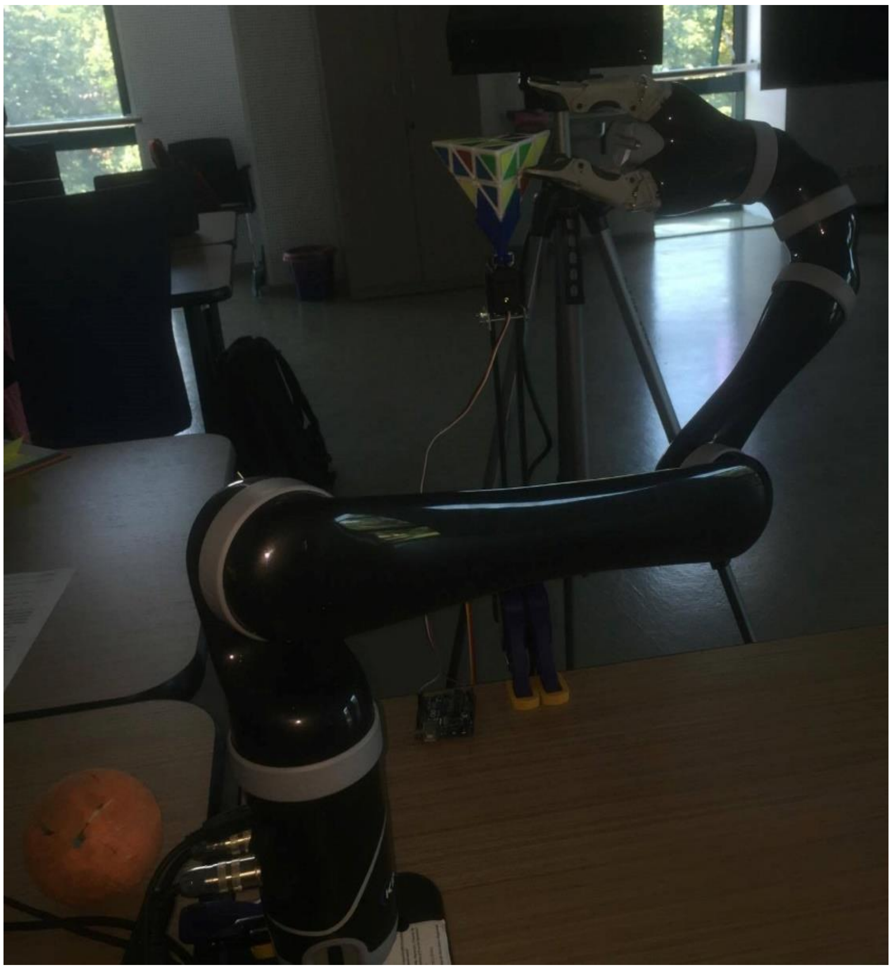


Figure 5.2 Move_BigCorner1

5.2.3 Move_BigCorner2

This move rotates the big corner placed on the right side of the robotic arm and it corresponds to Move 2. I’ve had applied the same principle as above and the values of the actuators are the following:

Table 5.3 Move_BigCorner2

Actuators	Final position (°)	Relative difference (°)
Actuator 1	176.86	101.85
Actuator 2	289.12	120.47
Actuator 3	258.25	202.39
Actuator 4	80.45	-167.25
Actuator 5	125.45	49.91
Actuator 6	184.45	108.82
Finger 1	4824	-
Finger 2	4830	-
Finger 3	4836	-

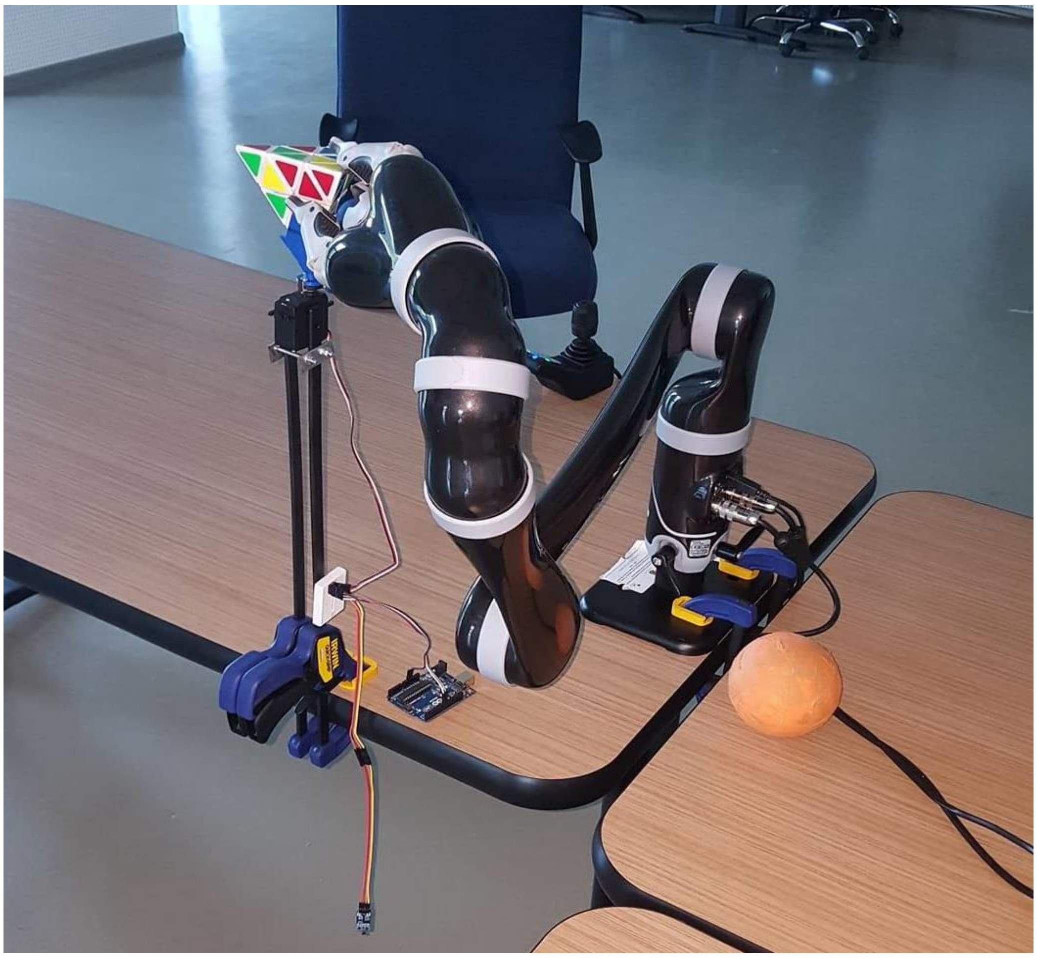


Figure 5.3 Move_BigCorner2

5.2.4 Move_BigCorner3

Starting from Home position, this move rotates the big corner placed in front of the robotic arm.

Table 5.4 Move_BigCorner3

Actuators	Final position (°)	Relative difference (°)
Actuator 1	255	-20.29
Actuator 2	169.69	2.3
Actuator 3	50.62	-6.5
Actuator 4	243.82	2.8
Actuator 5	68.18	-14.52
Actuator 6	15.82	-59.93
Finger 1	5184	-
Finger 2	5196	-
Finger 3	5148	-

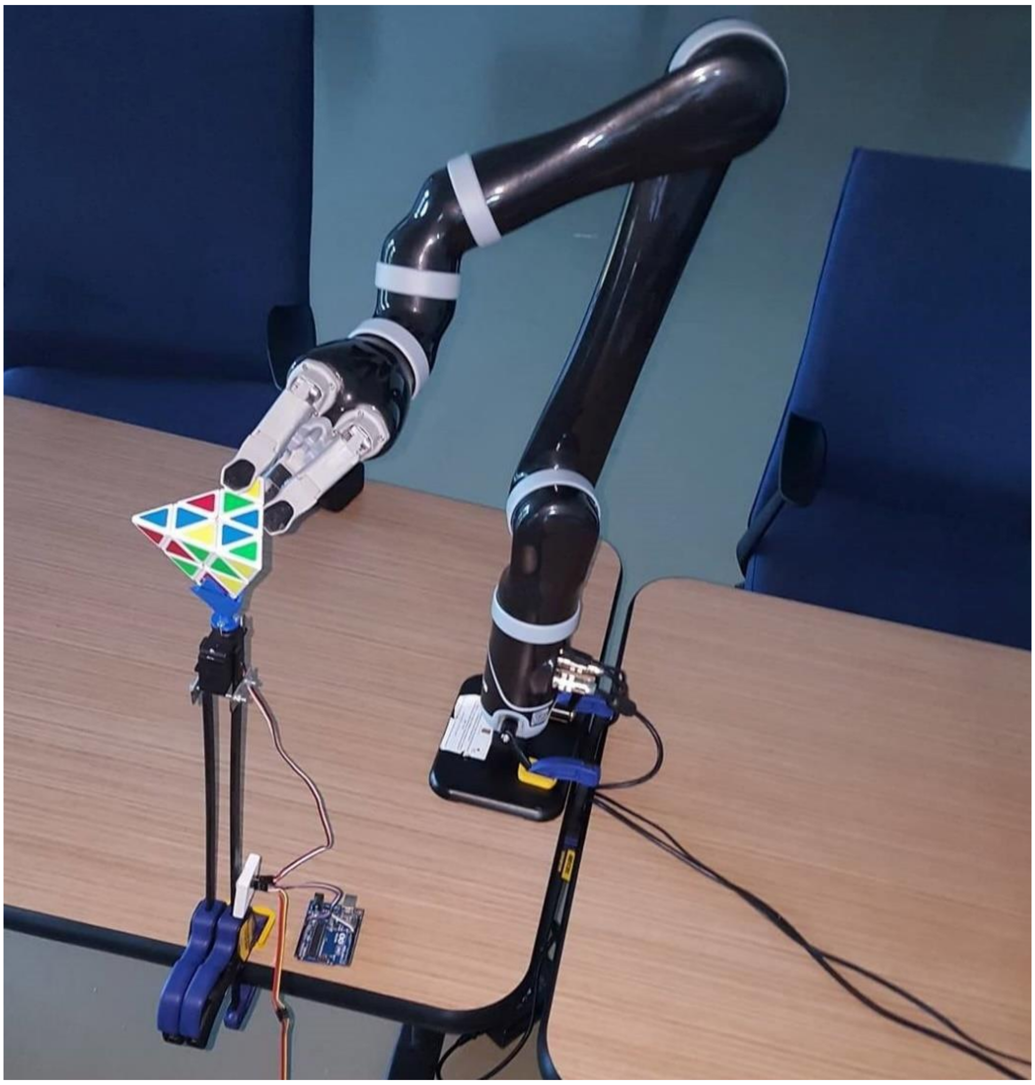


Figure 5.4 Move_BigCorner3

5.2.5 *Rotate_Layer*

This move corresponds with the 4th move from Chapter 4.

Table 5.5 Rotate_Layer

Actuators	Final position (°)	Relative difference (°)
Actuator 1	215.68	-1.63
Actuator 2	197.91	30.52
Actuator 3	122.32	65.19
Actuator 4	250.5	-9.8
Actuator 5	205.5	122.8
Actuator 6	177.68	101.93
Finger 1	3456	-
Finger 2	3420	-
Finger 3	3414	-

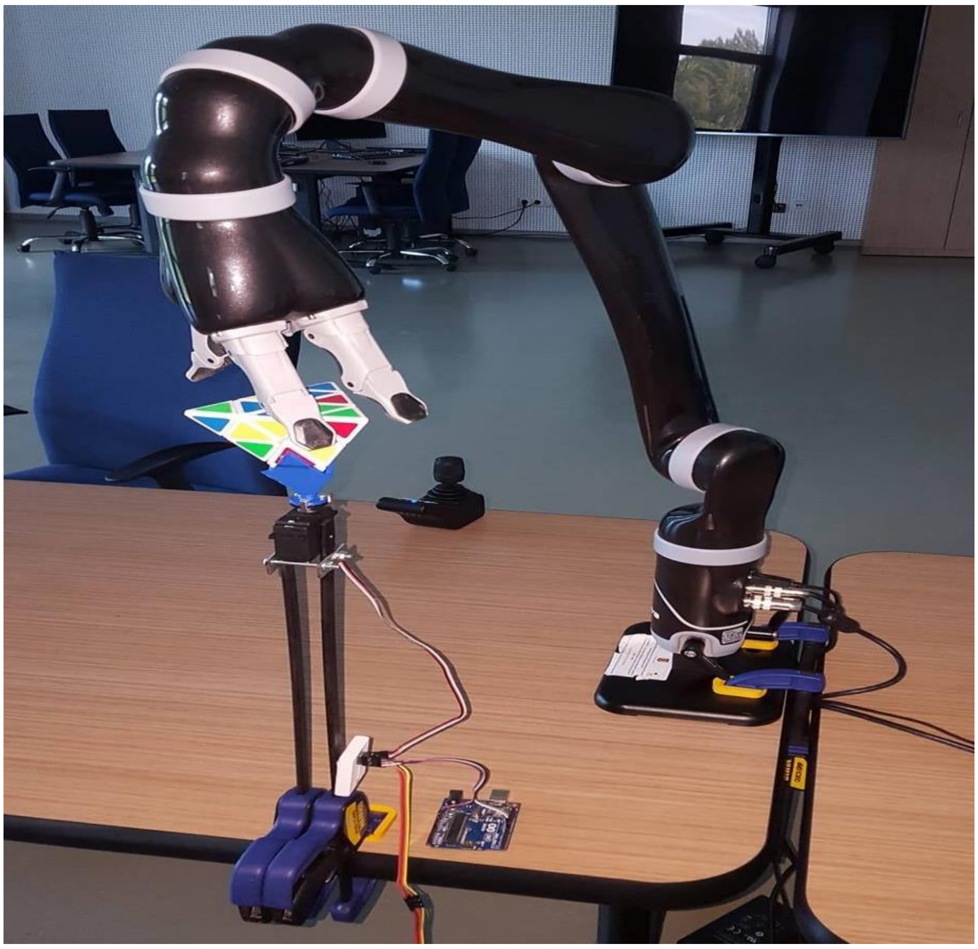


Figure 5.5 Rotate_Layer

5.2.6 *Move_SmallCorner*

Move_SmallCorner assures the rotation of the corner corresponding to Move 3. I chose this one because this is the vertex of the pyramid that faces the robotic arm. Rotating the corner is a very delicate movement, requiring high precision and dexterity, so it can be performed only from this position. To rotate the other corners, the arm will firstly perform Move 4 (once or twice depending on the case), so the corner is facing the robotic arm. Following, it rotates the corner and after that makes the reverse step, so the Pyraminx will remain in the same position.

Table 5.6 Move_SmallCorner

Actuators	Final position (°)	Relative difference (°)
Actuator 1	262.78	-12.21
Actuator 2	169.68	2.3
Actuator 3	50.68	-6.45
Actuator 4	243.82	2.8
Actuator 5	79.98	-2.72
Actuator 6	264.41	188.66
Finger 1	5988	-
Finger 2	6000	-
Finger 3	5700	-

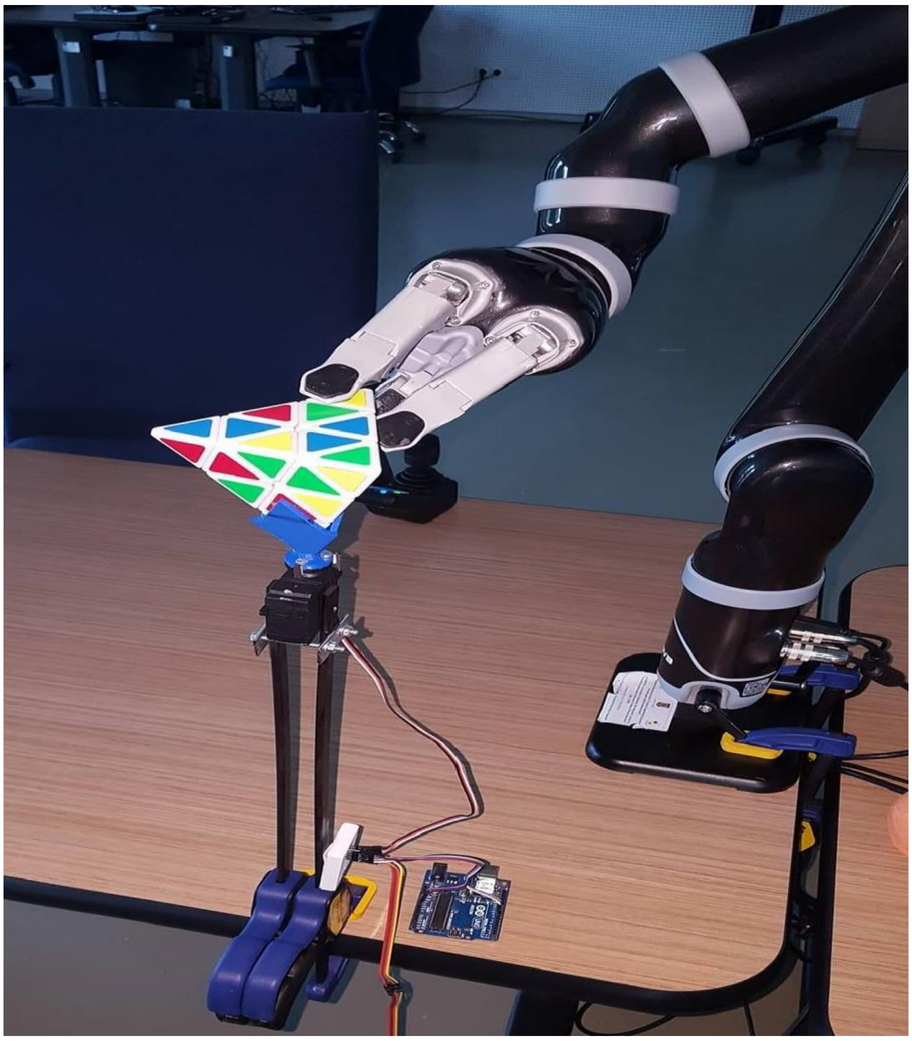


Figure 5.6 Move_SmallCorner

CHAPTER 6 CONCLUSIONS AND FUTURE STEPS

6.1 GENERAL CONCLUSIONS

The main objective of the thesis was to develop an autonomous system, capable to solve the Pyraminx puzzle using the robotic arm Jaco^2, created by Kinova Robotics. This project was composed of three main parts: data acquisition, Pyraminx solving algorithm and implementation of the moves on the robotic arm.

Starting from the premise mentioned in the introduction, that the main purpose of the ensemble described is to assist persons with multiple disabilities, I wanted to test the sensibility and the precision of the robotic arm. My test consisted in solving the Pyraminx and after developing the necessary logic and obtaining the wanted results, I was able to observe the high level of dexterity together with the behavior of the robot in this context.

At the current configuration, Jaco^2 may find its utility in the life of an autistic or a slower developing child by interacting with him. It would be able to maintain the attention of an ADHD child by its moves, conduct and practice. It can also play games aiming to develop the exchange with a special case as mentioned above.

In space, the articulated member had been found to be very handy for the Curiosity project as a manipulator. Like the Kinova product I've used, the space manipulator called Candarm has a place in the multi degree of freedom robotic arms.

The tasks performed by my system are a manifold of movements and data interpreting means. The decisions are taken based on the logic written entirely by me for each part of the aggregate. The images are captured based on the script designed for the Kinect. After this step, they are interpreted and the conduct of the robotic member is decided relying on the cases predicted in the dedicated code.

6.2 Personal Contributions

For this paper, my contributions are the following:

- I created the environmental setup so that the robotic arm may interact with the puzzle.
- I found a method to rotate the 3D-printed stand, hence the Kinect module has a fixed position.
- I developed the code for the image acquisition.
- I implemented two different methods to perform the color recognition task.
- I designed an algorithm that solves the Pyraminx based on four main moves.
- I programmed the robotic arm to execute the moves relying on the output of the solving algorithm.

6.3 Future Work

As the technology evolves day by day, a debutant project needs to keep the pass, or at least take into account the tendency oriented towards the artificial intelligence. That is why I am contouring the idea of creating a server where to deploy the logic for adaptive locomotion and learned information. By these, the arm should be able to identify and execute the fastest and targeted route to the desired object to be grabbed.

Another feature that I would like to add to the system is the vocal command, as another type of external stimuli. By adding sensors to the arm, it would be able to identify and avoid obstacles. This way, the arm will be safer to use by incapacitated persons, reducing the risk of accidental harming.

In a different context, let’s imagine 4 identical arms of this type, working together and forming a compact ensemble. It would be able to perform almost all the moves that a human being can execute with the same dexterity and precision. In addition, we would be able to choose the firmness of the grabbing, from a baby grip to a titanic clutch.

REFERENCES

[1] [Kinova] Kinova robotics website (http://www.kinovarobotics.com)

[2] [User Guide] Kinova robotic Jaco User Guide (http://www.kinovarobotics.com)

[3] [Monk, 2011] Monk, S., “Programming Arduino Getting Started with Sketches”, 2001, McGraw-Hills, 071784221

[4] [Nerd Paradise] Pyraminx Puzzle (http://www.nerdparadise.com/puzzles/pyraminx)

[5] [Ruwinx] Rubick’s Triangle (http://www.ruwix.com/twisty-puzzles/pyraminx-triangle-rubiks-cube)

[6] [Frey, 2001] Frey, A., Singmaster, D., “Handbook of Cubic Math”, 2001, Lutterworth Press, 0718825551

[7] [Beasley, 2006] Beasley, J.,” The Mathematics of Games”, 2006, Dover Publications, 0486449769

[8] [Online Encyclopedia]“Standard deviation”(https://en.wikipedia.org/wiki/Standard_deviation)

[9] [Open CV] “Canny Edge Detection” (http://docs.opencv.org/trunk/da/d22/tutorial_py_canny.html)

[10] [Datasheet] ”SENSITIVE HALL-EFFECT SWITCHES FOR HIGH-TEMPERATURE OPERATION”

[11] [Specifications] “DS04-NFC Servo 360-Degree Continuous Rotation Servo DC Gear Motor For Arduino/Raspberry-Pi/Robotics” (https://www.robomart.com/ds04-nfc-servo-360-degree-continuous-rotation-servos-dc-gear-motor)

[12] [User Guide] “Kinect for Windows SDK Programming Guide” (http://www.doc.flashrobotics.com/download/doc/Kinect%20for%20Windows%20SDK%20Programming%20Guide.pdf)