END-USER FRAMEWORK FOR ROBOT CONTROL

A Thesis

Presented to the Faculty of the Graduate School

of University of Texas at Arlington

in Partial Fulfillment of the Requirements for the Degree of

MS in Computer Science and Engineering

by

Kaustubh Rajpathak

May 2021

ABSTRACT

This thesis describes in detail a developed end-user framework for a human-robot collaborative system for common tasks, such as pick and place. The system is designed for semi-automated pick and place tasks as well as manual operation making it flexible for multiple use-case scenarios. The goal of the system is to make the robotic system multi-functional, easy to use with a graphical user interface and should perform common tasks with the help of a human teammate. Integration with object recognition neural networks (YOLOv3) and an RGB-Depth camera help automate pick and place tasks with a wide variety of objects.

ACKNOWLEDGEMENTS

I would like to thank Prof. Fillia Makedon and the Heracleia Lab at University of Texas. Arlington for providing me the golden opportunity for working on this project. I would also like to thank my supervisor, Dr. Maria Kyrarini for her invaluable guidance, providing essential information and support for completing my thesis and Dr. Nicholas Gans for accepting to serve as my committee member. This work has been partially supported by National Science Foundation grants IIS 1565328 and IIP 1719031. This material is based upon work by the author. Any opinions, findings, conclusions, and recommendations expressed in this paper are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF). I would like to thank Jackson Liller and Akilesh Rajavenkatanarayanan for assisting with the integration of the mobile base and the robotic arm. I would also like to thank Krishna Kodur for his help in developing drivers and required kernels for the system. Finally i would like to express my gratitude for my family and friends for providing invaluable support and love for pursuing my masters degree. Special mention to my mother Manjusha Rajpathak who is a master herself for guiding me towards a path to pursue higher education.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Ever since the industrial revolution came about, humans have longed to optimize manufacturing practices [19]. This thirst for optimizing work ways slowly extended to other disciplines such as surgery [10], search and rescue [2] [41], healthcare [28], logistics [45], service and assistive robots [52] [16]. One thing common between all these areas is that there are often repetitive and simple tasks involved which are commonly done by a human worker. Robots came about changing this scenario as they could do a lot more work done in such tasks than a comparable human worker. Initially robots were limited to a fixed operating pattern and a fixed location commonly known as ”cages” in the industrial automation domain. As the inclusion of robotics grew, there was a need for the robots to perform a diverse range of tasks and not being restricted to a constant environment.

Human-Robot Collaborative (cobot) applications started becoming more and more common as robots were freed from their cages. Cobot application reduce the effort for the human counterpart and improve task times [14]. There are some challenges which need to be addressed with cobot applications. The robots involved should be safe to operate in the same environment as the user, should offer easy to use control for the user and should react in real-time input [6]. These challenges were solved by introducing collision detection systems, intuitive user interfaces by using augmented reality, traditional computer based GUI's, gesture based control systems and voice based systems.

In further surveys conducted for the adoption of cobot applications, having a user interface which was easy to deploy, worked in a noisy and cluttered environment and was cost effective to deploy were preferred. This meant that using fancy hardware based user interfaces such as AR, VR [15], gesture based [35] [51] and voice based user interfaces [36] were only applicable in special use case scenarios. Having a computer based user interface was considered most generic for cobot applications.

A framework is proposed in this thesis that enables novice end-users to control a robotic system in an easy-to-use approach through a Graphical User Interface (GUI) from a remote computer. Adding useful automation systems, such as object detection and position estimation of objects, enable the user to efficiently perform common manipulation tasks. The robotic system consists of a robotic arm which is mounted on a mobile base.

1.1 Motivation

Robot automation is quickly becoming the norm in many industries such as manufacturing [19], logistics[45], healthcare [28], or even personal use robots [8]. There is a growing demand for robots and humanoids which stem from both intrigue and functional use of the machines. Along with the excitement in the adoption of these technologies, there also is a growing fear among labor unions across the globe fearing the loss of their jobs [11]. The proposed framework in this thesis not only helps existing workers adapt to new and better work environments but also helps the companies with a smooth transition to automation.

As we have shifted to automated systems, there always has been a fear of robots malfunctioning [17]. Having a human supervise the actions of the robot is often the chosen direction to ensure the operational security of the robots. Such an example can be seen in self-driving cars where the driver must be sitting in the driver's seat despite the car being perfectly capable of navigating through regular traffic situations. Having a human collaborate with a robot makes even more sense when we want the robot to perform a wide range of tasks.

Currently implemented Graphical User Interfaces (GUIs) for the most common industrial robots require the end-user to have robotic expertise [23]. Although existing software enables the end-user to program the robot in limitless ways, there often is a steep learning curve associated with using it and is prone to human error [1]. This makes the software complicated for the non-experts to use, discouraging wide spread adoption of robotic systems. Thus there is need for a GUI which makes it is easy to use for operating robotic systems in common world scenarios.

1.2 Objectives

The main objective of this thesis is to develop an intuitive and easy to use framework that would enable novice users to interact with a robotic system for basic pick and place tasks. To achieve this, a camera is required to enable the robot to "see" the environment. The robotic system is required to detect and recognize objects in the scene and the user provides instructions from a Graphical User Interface, which is easy to use. More specifically, the tasks of this thesis are defined as follows:

- Implement easy to use GUI for robot arm control.

- Add automation to common tasks such as pick and place.
- Implement effective motion control for mobile base.
- Integrate object detection with GUI.
- Evaluate usability of system.

1.3 Thesis Overview

This thesis is structured in the following chapters:

- Chapter 2 discusses the related work in building user interfaces for teleoperation and human-robot interaction.
- Chapter 3 explains the background concepts of ROS, Stereo Depth Imaging, Neural Networks and the MoveIt Framework which are needed to understand the implementation of the framework.
- Chapter 4 explains the proposed framework and the workings of its various components.
- Chapter 5, Evaluation details the evaluation methods used to measure the usability of the system.
- Chapter 6 summarizes the conclusions of the thesis and discusses new possible research from the presented work.

CHAPTER 2
RELATED WORK

In this chapter, we explore the different types of user interfaces used for teleoperation and robot control mechanisms for robotic systems. These user interfaces are differentiated by their use case, operating methods and the levels of autonomy. The user interface implemented in this thesis aims to be for a generic use case, controlled by a computer using button based graphical user interface and should provide complete manual control (teleoperation) and semi-autonomous operation for pick and place tasks.

2.1 User Interfaces for Service Robots

The primary function of these robotic systems is to be as a support for the elderly or people with disabilities. The user interfaces used in these type of systems need to be easily understandable by any user and should require minimum effort for completing a task. In [31], the author has proposed an user interface which is controlled by a handheld device. The systems is developed with three levels of autonomy. The first level is the fully autonomous mode which will be used by the elderly to instruct the robot for a task. The second level is the semi-autonomous mode where if the robot is not able to complete the task autonomously, it will call for assistance from caretakers. The third level is complete teleoperation mode which is meant for trained users to perform complex tasks.

Shwarz et al. [43] propose a similar system where the interface is divided in three levels of autonomy, body, skill and task control. This systems is meant to be used by a user who can operate a user interface from a remote computer. It adds a live camera image, external robot views and 2D laser scans to provide situational awareness to the user. The body level of autonomy is used when the robot recognises a completely automated task, the skill level is used to partially autonomous task such as moving the robot to a specific coordinate or grasping objects in the camera view. The task level is used when the user knows what operation is required with the respective values to complete the task. For example in task level autonomy the user has to select the move robot operation and then specify the coordinates.

Peppoloni et al. [35] implements a robot control interface in which the robot is controlled by tracking the movements of the user. The user′s movements are tracked using a leap motion device. The robot systems operates on two levels of autonomy, direct control and waypoint following. When using direct control, the robot mimics the hand movements of the user. The user can also control the end-effector by using his hand to make a grasp gesture. In waypoint following, the user defines certain positions for the robot to follow through. The robot will follow the trajectory which passes through the all the positions. In this system, the user receives feedback using a Kinect camera sensor and a head mounted display.

2.2 Industrial Robots

Industrial robots are commonly used in assembly lines of factories. The user interface requirements for this use case scenario are having high dexterity and time required to complete a task. Industrial robots are also used in isolation hence require two separate machines to achieve teleoperation. The host machine runs the user interface and the data is sent to the machine connected to the robot. To achieve real-time operation, latency issues must be taken care of in both the network and the host machine. Zhao et al. [51] implement a user interface similar to [35] where the robot mimics the actions of the human arm. In [35] the robot mainly follows the position of the hand and not necessarily in the orientation of the human arm. [51] use a human arm teleoperation which mimics the robotic arm in both the position and orientation of the human input gesture. The system they propose builds a model of the arm and calculates the rotation of each joint in the arm and moves the corresponding joints in the robotic arm by the same value. To adjust for size difference between the human and robot arm they use inverse kinematics which gives the joint angles needed for the robot arm to achieve the same displacement as the human arm.

An virtual reality based approach for controlling an industrial robot is detailed by Gonzalez et al. [15]. In an unique approach, the authors of this paper use a mini-robot arm to control the physical robot arm. The device used to control is the Phantom Mini Optic Device seen in Figure 2.1.

To give the user contextual awareness, an Amalgamated Vision (AV) head mounted display is used which run in the unity framework to provide the user with location and depth information of the environment. To include the depth information for the user, the authors use a trio of 3D cameras to eliminate blind spots for the user and provide a immersive experience as if the user is operating the robot from the shop floor. A force sensor is also attached to the end-effector of the robot to give the user feedback on how tight the grasp is. Haptics of the robot motion are provided through the phantom mini optic device. The authors

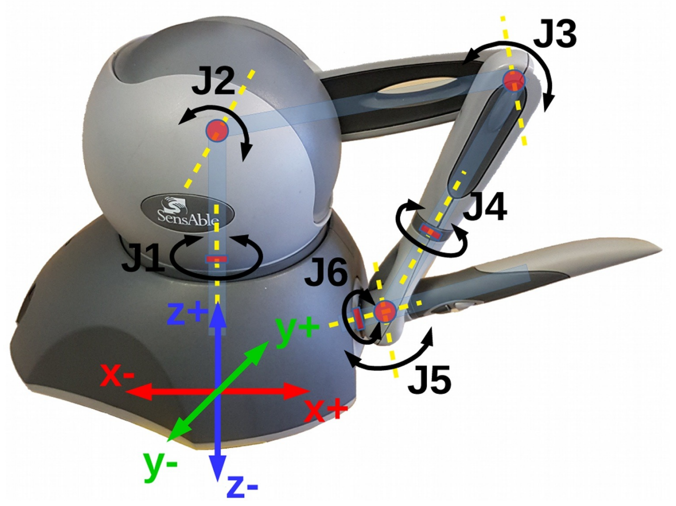


Figure 2.1: Phantom Mini Optic Device

do not mention any separate autonomy modes which means that the system purely runs as a teleoperation interface.

Another approach to control a robot was by using voice commands as demonstrated by Poncela et al. [36]. Speech recognition was implemented using the Julius/Julian voice decoder. The user can say a command such as move 1 meter, and the robot arm will move 2 meters linearly in is current orientation. This voice decoder in implemented in spanish. There is no support for any other language. This system only works in teleoperation mode as automation requires complex commands which the voice decoder has difficulty to parse it to commands.

2.3 Manufacturer Specific User Interfaces

Some robots come with a user interface with the robot to get the user started in using the robot. They are often built to provide semi-autonomous motion controls. The user interface is focused more on achieving a large variety of tasks as the manufacturer wants the robot to be used in a wide variety of scenarios.

2.3.1 Franka Emika Desk

Franka Emika Desk is a web interface which enables user to easily program the robot to perform various tasks. The GUI provides the user to program the robot using block based actions. For example, if an user wants to perform a pick and place task, he needs to define a Cartesian movement block, End-effector movement block, Post grasp Cartesian movement block and an end-effector movement block. This process needs to repeated for every object which needs to be picked up and placed.

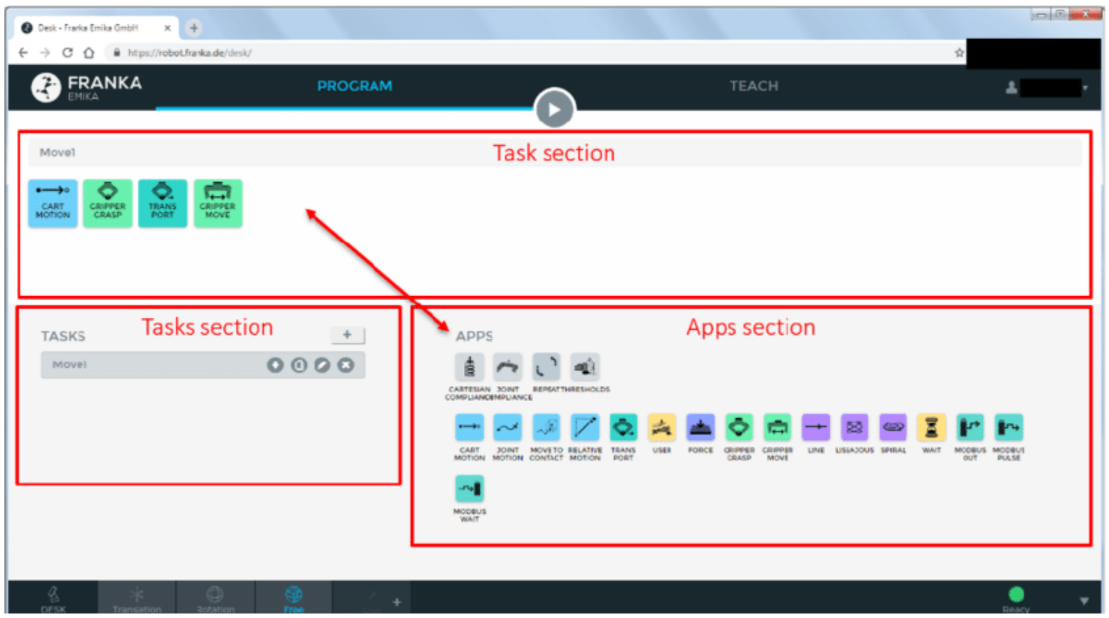
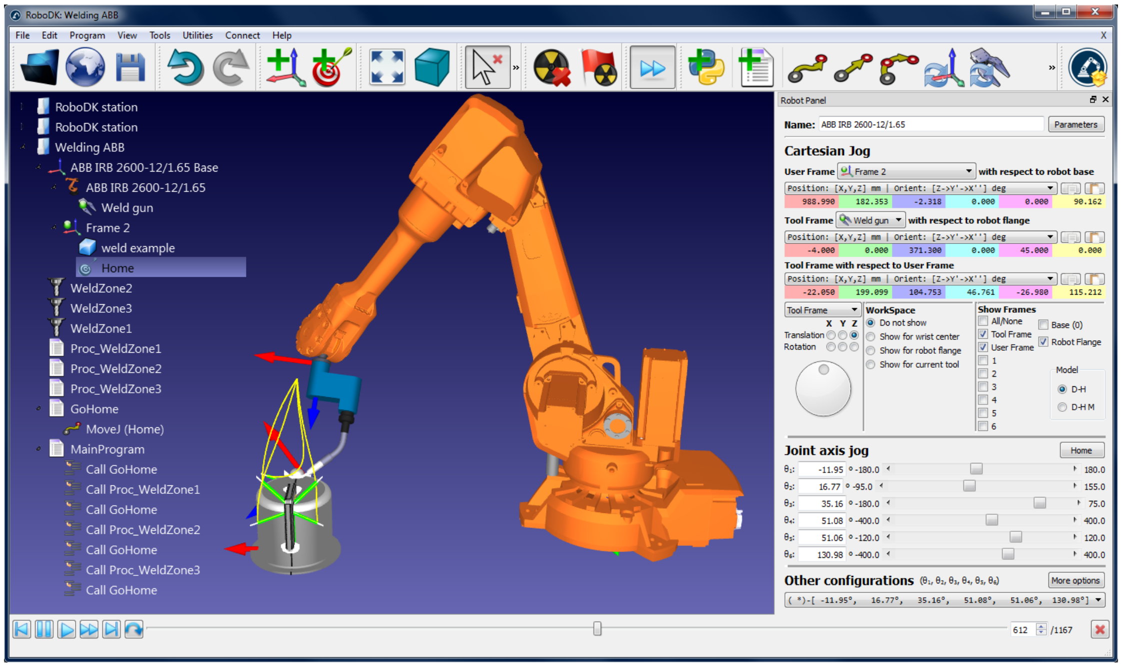


Figure 2.2: Franka Emika Desk [22]

This process as you can see is lengthy and not viable for a dynamic environment. The user also needs to have knowledge of every block and what information is needed for every block. This GUI is also only useful if the robot and the objects are at a fixed position. The software also does not provide functionality to add external sensors such as a camera. The software also doesn't support adding another robot such as a mobile base to make a multi-robot robotic system. This software is also useful for a specific robot, the Franka Emika Panda.

2.3.2 RoboDK

RoboDK [32] is a software which is exclusively used by robots manufactured by Universal Robots. This is a highly functional software which is used in an industrial setting where the entire pick and place is automated. The automation steps are programmed as fixed tasks which run in a loop as long as the robot is powered.



Name: ABB IRB 2600-12/1.65

Cartesian Jog

User Frame: Frame 2 with respect to robot base

```
Position: [X,Y,Z] mm | Orient: [Z->Y'->X''] deg
988.990   182.353   -2.318   0.000   0.000   90.162
```

Tool Frame: Weld gun with respect to robot flange

```
Position: [X,Y,Z] mm | Orient: [Z->Y'->X''] deg
-4.000   0.000   371.300   0.000   45.000   0.000
```

Tool Frame with respect to User Frame

```
Position: [X,Y,Z] mm | Orient: [Z->Y'->X''] deg
-22.050   199.099   104.753   46.761   -26.980   115.212
```

Tool Frame — Translation: X Y [✓ Z]; Rotation: X Y Z

WorkSpace: [✓ Do not show] Show for wrist center Show for robot flange Show for current tool

Show Frames: All/None [✓ Tool Frame] [✓ User Frame] 1 2 3 4 5 6 Base (0) [✓ Robot Flange]

Model: [✓ D-H] D-H M

Joint axis jog

θ_1:	-11.95 °	-180.0	180.0
θ_2:	16.77 °	-95.0	155.0
θ_3:	35.16 °	-180.0	75.0
θ_4:	51.08 °	-400.0	400.0
θ_5:	51.06 °	-120.0	120.0
θ_6:	130.98 °	-400.0	400.0

Other configurations (θ_1, θ_2, θ_3, θ_4, θ_5, θ_6)

```
( *)-[ -11.95°,  16.77°,  35.16°,  51.08°,  51.06°,  130.98°]
```

612 /1167

Figure 2.3: RoboDK Software [32]

The software requires user to know knowledge of the robot to program it.

Once programmed the robot will perform a fixed task until it is reprogrammed which makes it unsuitable for dynamic environments. It also depends on the CAD file of your robotic system, preventing it from using it in multi-robotic systems. Since there is no integration available for camera systems, objects cannot be tracked hence have to kept at a fixed location.

2.4 Robot Learning by Demonstration

Learn by demonstration is a method of programming the robot by manually teaching the robot to do a certain task. Once the robot learns a certain task it will repeat it on its own without human supervision. It is quicker than pre-programming as the user doesn't need to have knowledge of the robot. To teach the robot, the user only has to move the robot to positions of pick and place.

Learning by Demonstration can be divided into two approaches by Argall et al. [3] and by Billard et al. [7] namely low-level skill learning at trajectory level (or low-level motor control learning) and high-level symbolic task learning (or high-level symbolic reasoning).

2.4.1 High-Level Symbolic Task Learning

In high-level symbolic task learning, the task is encoded according to the sequences of predefined actions. This approach allows the robot to learn the sequence of actions, so the robot can learn high-level tasks [7]. It relies on a priori knowledge to be able to abstract the important key-points of the demonstrated task [27]. For eg, to perform a pick and place task using this method, the robot learns the following steps:

- Identify object to pick.
- Move to pre-grasp pose.
- Grasp object.
- Move to place position.
- Return to home pose.

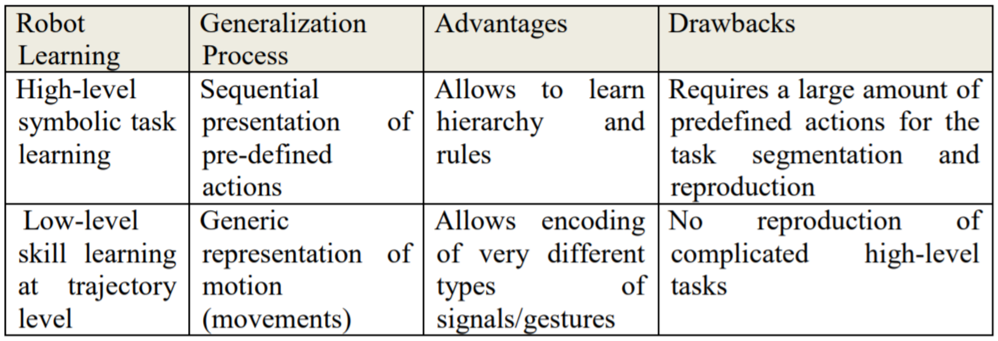
2.4.2 Low-Level Motor Control Learning

In low-level motor control learning, the robot learns the orientations of the joints required to perform a certain motion. This approach allows the robot to learn a sequence of poses, so the robot can perform movements. This approach does not allow the robot to learn high level tasks such as pick and place as it is only limited to learning the motion of the robot arm and not the sensors involved to identify objects. A typical motion pipeline for a low-level motor control learning system can be seen in industrial applications where the robot moves in fixed trajectories.

The two approaches can be summarized in following table 2.1:

2.5 Robot Control Mechanisms

In this section. we discuss various ways of controlling a robot. We discuss the merits and de-merits of each of the approaches to find how they are useful in



Robot Learning	Generalization Process	Advantages	Drawbacks
High-level symbolic task learning	Sequential presentation of pre-defined actions	Allows to learn hierarchy and rules	Requires a large amount of predefined actions for the task segmentation and reproduction
Low-level skill learning at trajectory level	Generic representation of motion (movements)	Allows encoding of very different types of signals/gestures	No reproduction of complicated high-level tasks

Table 2.1: Comparison of High Level and Low Level Learning [26]

different scenarios. All the methods of robot control mentioned in this section are used in our framework.

2.5.1 Torque Control

Every joint of the robot can be controlled by specifying the rotation angle the joint needs to rotate. This method of control is commonly known as torque control. Such controllers are common in low-level robot controllers which communicate with the motors of the robot at a physical level. These controllers can be communicated with from a GUI but the usage of these is not intuitive as there are many joints in the robotic and their combined rotation produces a simple motion which the user cannot intuitively control.

2.5.2 Motion Planning

Motion Planning is a path solving algorithm to move the robot from start to destination coordinates. The movements of the robot can be achieved by specifying a destination coordinate from the GUI. The motion planning algorithm returns back a set of configurations which is passed to the low-level controlled to achieve motion. A configuration consists of the rotation angles of every joint for the robot to move from start to end coordinate. The coordinate system is much easier to understand for the user than torque control as there is a direct correlation from coordinate axes to real-world motion of the robot. For example +0.1 on y axis will move the robot arm upwards by 10 cm. It is still difficult for the user to identify exact coordinates of an object which needs to be picked as the user is not familiar with coordinate frame of references. This can be solved by implementing position estimation of objects using a camera preferably with a depth sensor.

2.5.3 Velocity Control

Velocity Control is a common control method for mobile bases. In this a speed value is set for the robot in a particular direction. For example 0.5 m/s in the x axis. This method is easy for the user to use as there is a direct correlation between input and motion. For a user not familiar with velocities, the GUI can implement velocity control similar to a car game by providing control using keyboard arrow keys.

CHAPTER 3

BACKGROUND

In the previous chapter, various systems of robot control such as GUI's and programming methods were explored. From those developing a GUI which implemented motion planning and velocity control in an intuitive manner is chosen for the framework. These methods were chosen as they are well suited for dynamically changing environments and their ability to move the robot in any direction with ease. To simplify motion planning, object position estimation is also implemented. Velocity control is implemented using keyboard keys. In this chapter, various concepts are explained which will help in understanding the system architecture explained in the next chapter.

3.1 Robots

This section describes the different robots used for building the free moving robotic system. A robotic arm is used for handling the objects for pick and place and a mobile base is used as a mounting platform for the robotic arm so that it can be moved around.

3.1.1 Robotic Arms

A robotic arm is a type of mechanical arm, usually programmable, with similar functions to a human arm; the arm may be the sum total of the mechanism or may be part of a more complex robotic system. The links of such a manipulator are connected by joints allowing either rotational motion (such as in an articulated robot) or translational (linear) displacement. The robot we are using uses rotational motion for its joints. Since the arm used has more than 3 joints, it is classified as an articulated robot.

3.1.2 Mobile Bases

A mobile base is a type of robot which moves using wheels commonly attached to the bottom of its chasis. A mobile base can be used to add movement to a static robotic system by mounting another robot on top of it. Mobile bases can vary by differing load carrying capacities and their ability to function in differing environments. Their movement is restricted to a certain plane of operation, meaning they cannot jump or duck.

3.2 Robot Operating System (ROS)

Robot Operating System, despite what the name would suggest is not an operating system but a collection of useful software tools built for robot development [47]. ROS implements a distributed architecture for easily accommodating sensors and motors to the robotic software. ROS divides the various components in the system into ROS nodes which can communicate with each other. For eg, The camera could have its own node and it could send the video to the neural network node for object detection. To understand how such processes work we need to understand the system architecture of ROS. The ROS architecture mainly consists of the following components: -

- Master
- Packages
- Node
- Publisher
- Subscriber
- Service
- Message

In the next sections, the ROS components are breifly explained.

3.2.1 ROS Packages

ROS Packages are folders which contain the ROS software you want to write. You can use pre-existing packages for your application or build one for you specific application. A ROS Package is organized in the following manner: -

```
catkin workspace/src
└─ package name
   ├─ package.xml
   ├─ CmakeLists.txt
   ├─ src
   │  └─ Python/C++ Files
   └─ include
      └─ Include Files for Python/C++ files in src folder
```

Figure 3.1: ROS Package File System

package.xml file describes the package and it's dependencies. CmakeList.txt file finds other required packages and messages/services/actions.

3.2.2 ROS Node

Each component in a ROS framewok is called as a ROS Node. A Node is essentially a process which communicates with other nodes through topics/services/actions. A ROS Node consists of publisher and subscribers. Publishers send data on topics and subscribers accept the data from the topic and process it. A ROS Node can have multiple publishers and subscribers.

3.2.3 ROS Master

A ROS Master is the master node for all ROS Nodes. There is only one master node per system. Every node is a slave to the master node. As with all distributed systems, the master node allocates separate threads for each ROS node running. The ROS Master acts like a server for all ROS services and actions. The slave nodes are clients which can communicate using pre-defined topics with the server. A ROS topic does not have a pre-defined topic, therefore the ROS Master acts as an intermediary between nodes so that the topics created by other nodes are accessible every node connected to the master. A ROS Node can create a topic which is kept track of the ROS Master Node.

3.2.4 ROS Publisher

A ROS Publisher is a process within a node which is used to publish messages on a topic. The publishers implemented in the robotic system are a camera image publisher, the neural network publisher, the summit publisher and move

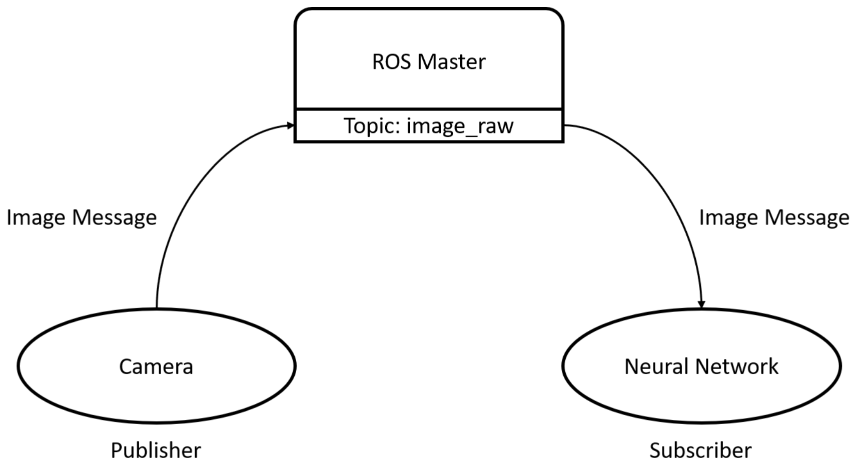


Figure 3.2: ROS Master with Camera Node and Neural Network Node

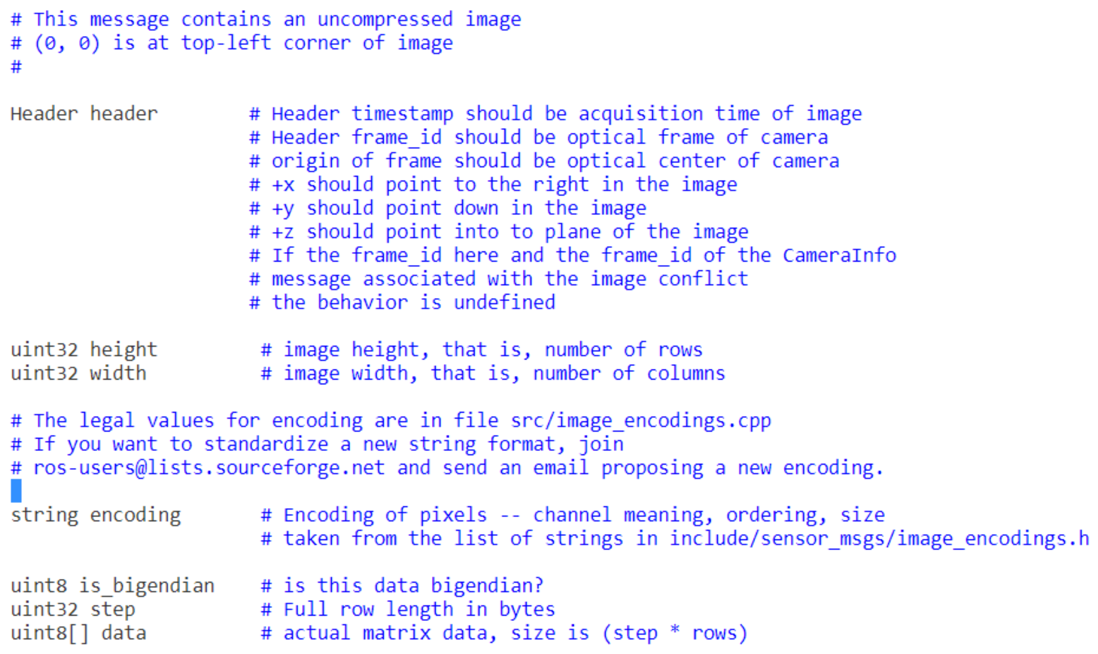
group publisher for visualizations on rviz. A ROS Publisher can create a topic to publish any data which is formatted as a ROS Message.

3.2.5 ROS Subscriber

A ROS Subscriber is a process within a node which accepts information published on a ROS Topic by a publisher. The subscribers implemented in the robotic system is the neural network subscriber which processes the information of the objects location. The subscriber calls a function ″callback″ as soon as a new message is published on the topic. The implementation of this function is dependant on the developer. The message passed on the topic can be processed by passing as an argument to the callback function.

3.2.6 ROS Messages

ROS Messages are essentially struct type data structures which ROS defines. Each publisher and subscriber has a unique message type that it can send on a topic or process. Some of the messages used in our system are a bounding box message, image message.



```
# This message contains an uncompressed image
# (0, 0) is at top-left corner of image
#

Header header        # Header timestamp should be acquisition time of image
                     # Header frame_id should be optical frame of camera
                     # origin of frame should be optical center of camera
                     # +x should point to the right in the image
                     # +y should point down in the image
                     # +z should point into to plane of the image
                     # If the frame_id here and the frame_id of the CameraInfo
                     # message associated with the image conflict
                     # the behavior is undefined

uint32 height         # image height, that is, number of rows
uint32 width          # image width, that is, number of columns

# The legal values for encoding are in file src/image_encodings.cpp
# If you want to standardize a new string format, join
# ros-users@lists.sourceforge.net and send an email proposing a new encoding.

string encoding       # Encoding of pixels -- channel meaning, ordering, size
                      # taken from the list of strings in include/sensor_msgs/image_encodings.h

uint8 is_bigendian    # is this data bigendian?
uint32 step           # Full row length in bytes
uint8[] data          # actual matrix data, size is (step * rows)
```

Figure 3.3: Image message [47]

Figure 3.3 shows the structure of a image message which is formed after processing the image frame output from the camera.

3.3 ROS Transform

ROS Transform is a tool which is used to transform coordinates from one frame of reference to another. Commonly in robotic arms, there are a lot of joints with separate axes and keeping track of each of them becomes a hassle as each one of them changes with movement. Using ROS Transform can help deal with this issue, especially when you have separate sensors like a camera in my case whose coordinate frames need to be transformed to the panda's frame of reference. The transformations used in the system are origin shift transformation and rotation transforms.

3.4 MoveIt Framework

MoveIt is a motion planning framework used to communicate with the robot [9]. Motion planning is a computational problem to find a sequence of valid configurations that moves the object from the source to destination. The term is used in computational geometry, computer animation, robotics and computer games. Although there are controllers provided for the robot which work with ROS, using those is a hassle as they do not support moving the robot using coordinates. MoveIt is also useful is it implements collision detection and gives us libraries which work in python instead of C++. MoveIt implements a convenient move group interface built on top of ROS to give high level abstraction of movement controls.

3.5 RGB-Depth Cameras

RGB-Depth Cameras are a specific type of depth sensing devices that work in an association with a RGB camera, that are able to augment the conventional image with depth information (related with the distance to the sensor) in a per-pixel basis. This type of camera is used in our system to get the position of the object in the real world. There are a few ways the camera can calculate the depth of an object.

3.5.1 Pattern Matching Depth Imaging

In this approach of depth calculation, the camera emits a pattern of infra-red light and observes the change in the pattern. If an object has a pattern projected on it, it will curve the light around it hence changing the pattern. An example of this can be seen in Figure 3.4.

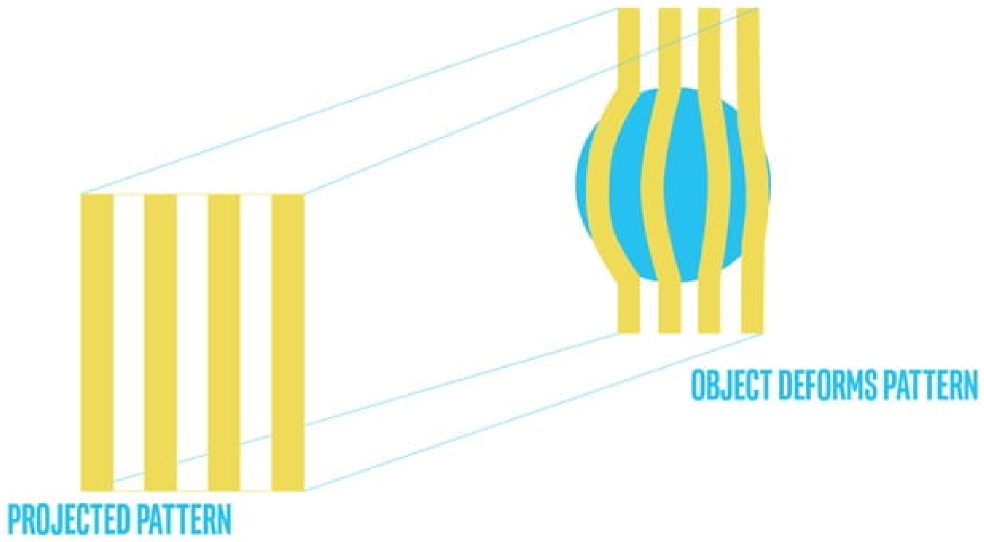


Figure 3.4: Structured and Coded Light [21]

This approach has some drawbacks. It cannot be used in an environment with lot of disturbance. Any other light falling on the object changes the IR Pattern projected. For the same reason, multiple camera setups cannot be used with such a depth imaging technique as the patterns projected from other cameras interfere with each other.

3.5.2 Stereo Depth Imaging

In this approach, two cameras are used in a stereo configuration like the human eyes. The working principal of this approach is the same as used by our human eyes. Whenever we see an object with two eyes, each eye sees a different version of the object, either shifted to the left or right depending on the eye you see it from. These two versions are combined in our brains and triangulation is used to calculate the depth of the object. This phenomenon is demonstrated in the Figure 3.5.

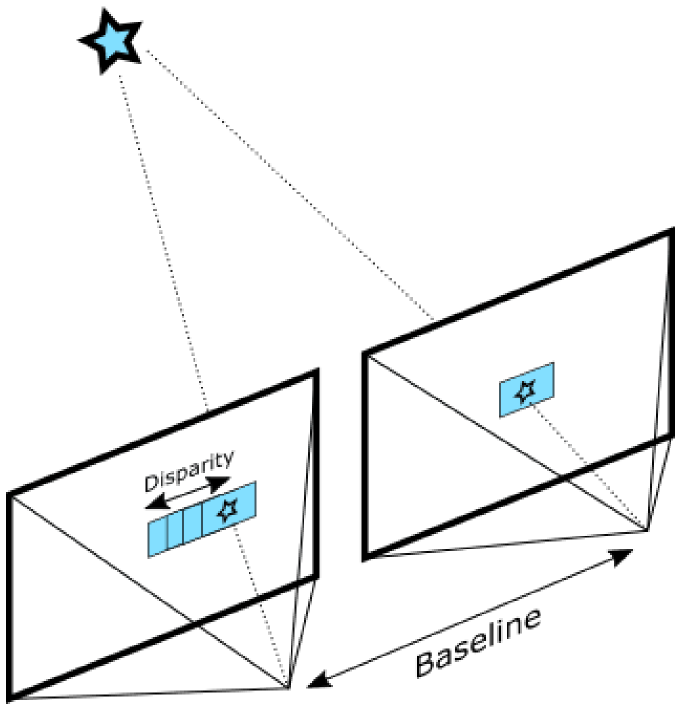


Figure 3.5: Stereo Depth Vision [44]

3.5.3 LiDAR Imaging

LiDAR stands for Light Detection and Ranging. As the name suggests, it uses light to calculate the depth of an object. It leverages the fact that the speed of light is known. A LiDAR scanner bounces the light of the object and measures the time required for it to return. Instead of using normal light, LiDAR's use lasers to avoid diffraction and loss of signal in the environment. As was the case with pattern imaging, LiDAR depends on the object and environment for it to receive the reflected light. This makes it unreliable in environments with lot of light such as outdoors. It also struggles with other LiDAR scanners in the environment as lasers from other cameras might be captured. Figure 3.6 shows the working principal of LiDAR.

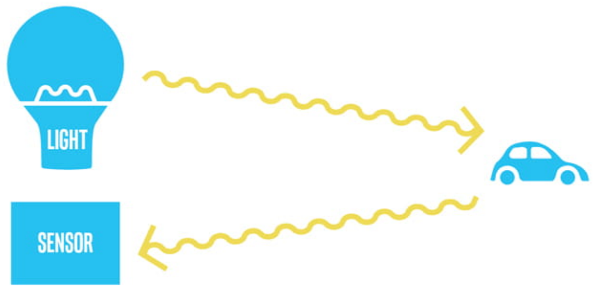


Figure 3.6: LiDAR Working Principal [21]

3.6 Deep Learning

Deep Learning is a part of machine learning which aims to solve a given problem by mimicking the problem solving methods used by our brains. Our brains consist of a interconnected layer of cells known as neurons which fire in a certain pattern to process information. Deep Learning uses a similar approach by

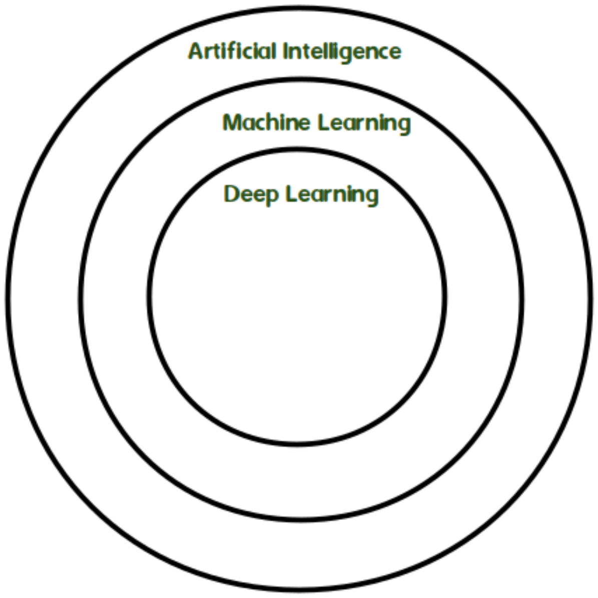


Figure 3.7: Deep Learning [42]

building a network of neurons. Each neuron has a certain value after which it ”activates” passing along its value to neurons it is connected to. The final layer of neurons is used for classification of the data. For e.g, to classify hand written digits, the final layer will consist of 10 neurons each corresponding to digits 0-9. If the 4th neuron is activated, 4 was the digit in the image provided as input.

Unlike machine learning, the problem to be solved is not divided into smaller tasks but solved as a whole. For e.g, to do image classification with traditional machine learning, the image must be first segmented to isolate the objects from the background, and then individual objects must be separated from one another using clustering and then finally shape matching can be applied. On the other using a neural network eliminates all these stages and simplifies the problem to accept input image as an array and output the class of the image. Deep learning is also able to identify complex relations in data such as contextual information. This does mean it takes longer for the model to train but testing data is fast.

3.6.1 Neural Networks

Deep Learning is implemented by a neural network. As the name suggests, a neural network is a network of neurons. In this section, the most basic implementation of a neural network the multi-layer perceptron is explained. The neurons are arranged in layers known as the input layer, hidden layer and output layer. The general design of a Multi-Layer Perceptron (MLP) for classifying hand written digits is shown in Figure 3.8.

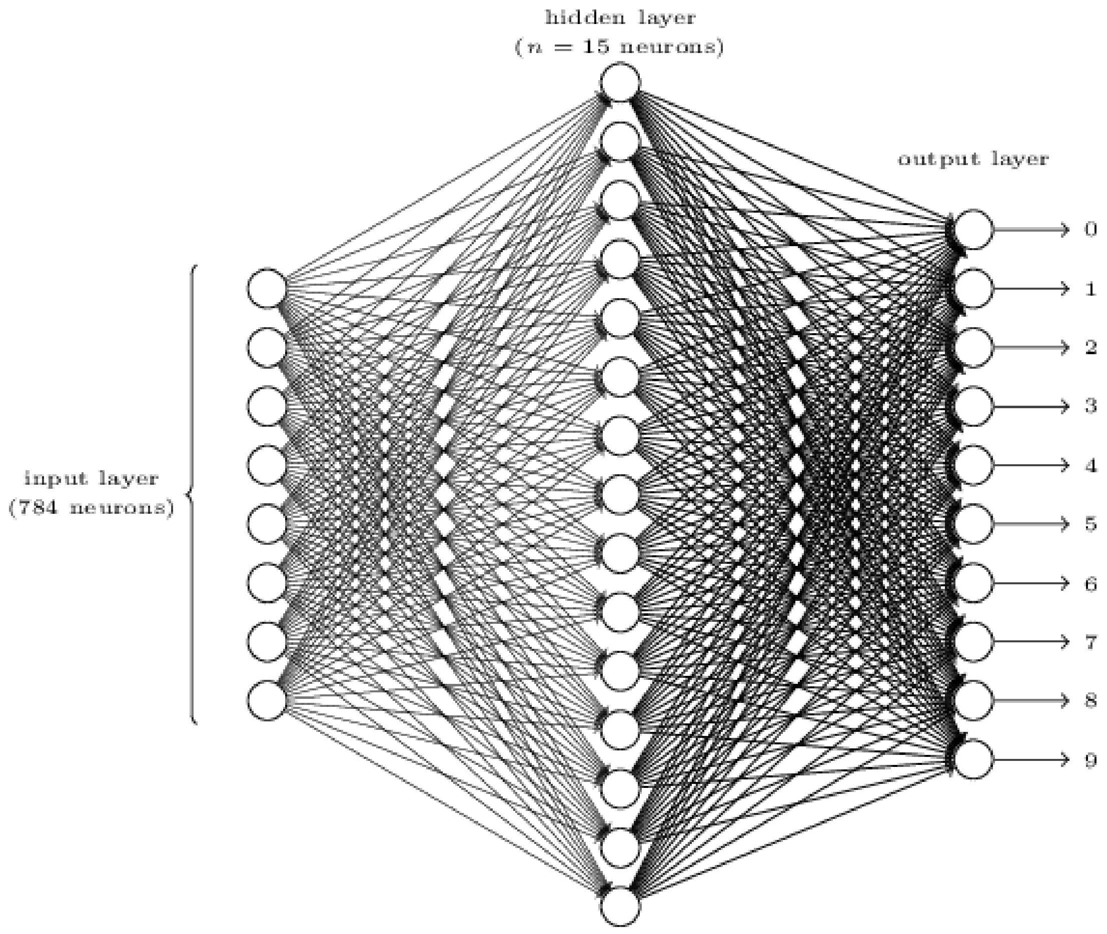


Figure 3.8: Multi-Layer Perceptron

Each neuron except ones part of the input layer has a bias value and an activation function associated with it. Every connection in the network has a weight. An activation function transforms the input into a fixed range output. A common example is a sigmoid function 3.1 which gives the output between 0 and 1. An activation function is needed to add non-linearity to the model. This is useful to identify complex patterns in the data. Without the activation function, the output will be a direct linear relation of the input. The entire model will act like a linear regression model with limited learning power.

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{3.1}$$

$$z \in \zeta \tag{3.2}$$

When the network is initialized it has random weights and biases. All the weights and biases must be adjusted for the network to produce meaningful results. This is known as training the network. For a supervised neural network, training is done with the help of a loss function. A loss function calculates the error in the output. A common loss function is the Mean Square Error (MSE) function as shown in equation 3.3. The goal of training the network is to minimize this loss function so that the predicted output is as close to the actual output.

$$MSE = \sum_{i=1}^{N} (x_a - x_p)^2 \tag{3.3}$$

where x_a is the actual output, x_p is the predicted output and N is the total number of neurons in the output layer.

To minimize the cost function, a technique known as *gradient descent* is used. To understand gradient descent lets take a simple example. Lets say we have a class of students. The teacher of the class wants the average of the class at a certain value for example 85% at the end of the term. The class is not an honest class, therefore every student copies from a bunch of students. After the first test in the term was graded a student scored 50% which was the lowest in class. Due to this student, the class average dropped considerably. The teacher being smart, focuses on bringing the score of this student higher than somebody who has scored 80% as him improving will have a greater effect on the average than the one who scored 80%. Subsequently to increase his own score, the student asks the other students from which he copied to score better than last time (An assumption that the student improves on the next test is made). The teacher repeats this process with every lowest scoring student per test until eventually by end of term, the class average is near 85%. This process of iteratively moving towards the steepest descent as defined by the negative of the gradient is known an gradient descent. In the example, the steepest descent was the student scoring the lowest in each test thus having a largest difference with the target average value.

In a neural network, to minimize the cost function we adjust the parameters (weight and bias) of the output neuron which is most away from the actual value. Doing this makes it modify the parameters of the neurons which are connected to it as the student did by asking the other students from which he cpopied from to perform better. This process is known as **back propagation**. This process is repeated for every input data until all the parameters of the neurons are adjusted to produce the lowest MSE. To train the model effectively, large quantity of input data with variety is required to produce a network which is not biased towards a particular input.

3.6.2 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a modification to the traditional MLP described in the previous section. One application of the CNN is image classification. To understand why CNN is used we must look at the drawbacks of an MLP when dealing with images.

Large Input Layer

An MLP's input layer consists of one neuron for each pixel in the image. Due to this, for high resolution images, the number of weights and biases becomes too large to train the model effectively in a reasonable time.

Loss of spatial information

The input layer flattens the image for input to the MLP. Commonly the neighboring pixel's are useful in identifying the image. Flattening the image causes the neighboring pixels to drift away from each other leading in lower accuracy.

Erroneous classification of shifted image of input

A shifted image is a variant of the image where the subject is moved to a different part of the image. With sufficient training the MLP expects the subject in the image at a specific region in the input image. When the subject is not in the same region, the network fails to identify the image.

A CNN fixes the drawbacks of an MLP by reducing the size of the image by using convolution and pooling operations. It reduces the size of the image so that the MLP which is used after the convolution and pooling operations is much smaller in size resulting in quicker and efficient training.

CHAPTER 4

PROPOSED SYSTEM

This chapter provides detail information of the proposed end-user framework and its functionality. This chapter also includes information about the hardware setup of the proposed system.

4.1 Hardware Setup

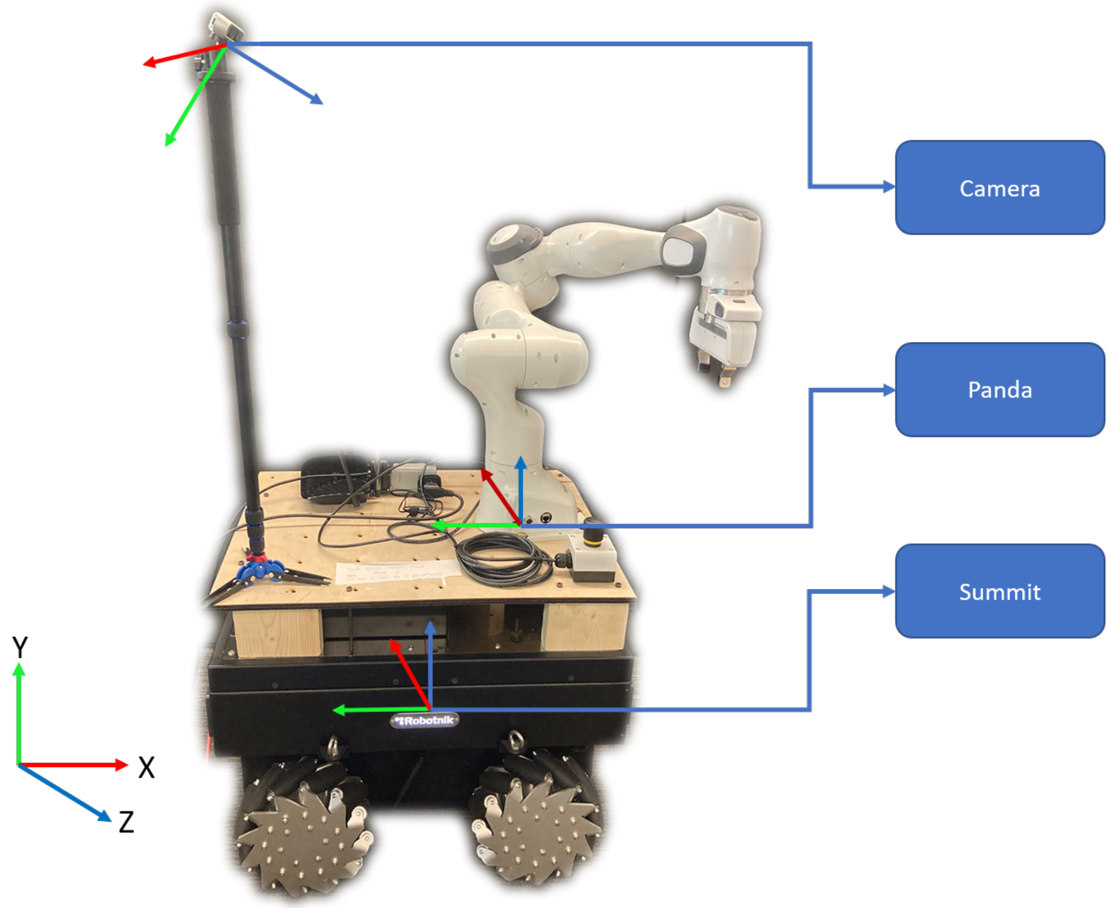


Figure 4.1: Robotic System

In Figure 4.1 we can see how the robotic system in real life. The robotic arm and camera are mounted on top of the mobile base. This setup allows for free motion of the system without interfering with the camera view for object detection. The respective coordinate frames are also shown for each component. In the next sections, these components are discussed in detail.

4.1.1 Franka Emika Panda

The robotic arm we are using is the Franka Emika Panda robot arm. We chose this robot because of its torque-driven characteristics that can detect the smallest of collisions making it safe in collaborative environments. The panda is also faster and precise to 0.1mm. The safe operative nature, low cost, precise and fast speed make it an ideal product for industry-wide cobot (collaborative robot) applications [13].

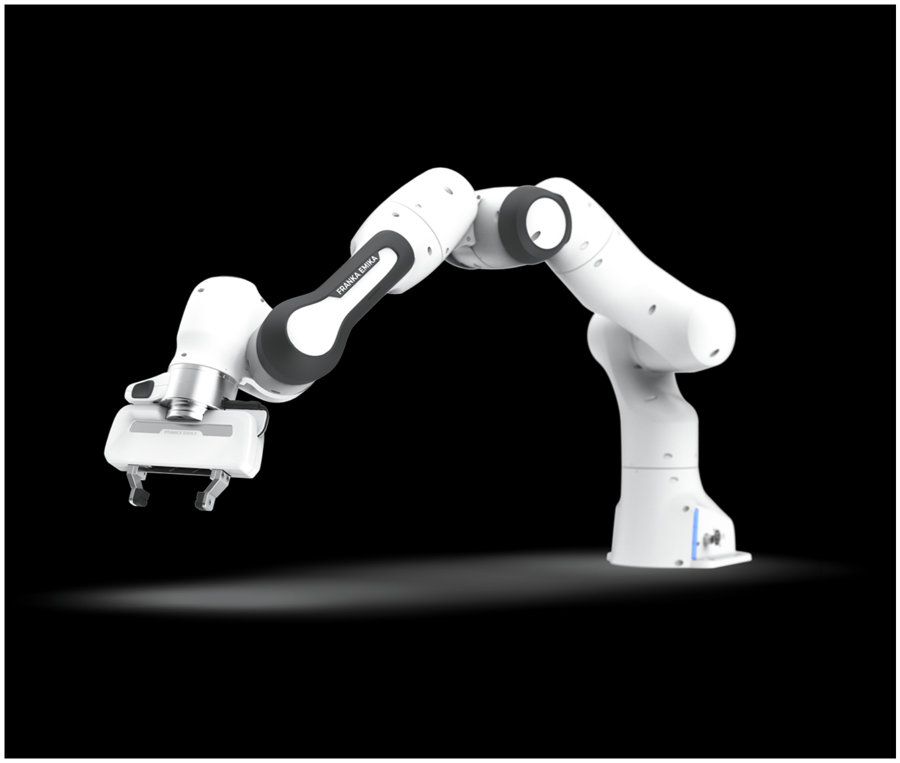


Figure 4.2: Franka Emika Panda

4.1.2 Robotnik Summit XL Steel

SUMMIT-XL STEEL is a robotic platform for application development (logistics, indoor transport, etc.). It has a robust design and can carry up to 250 kg payload. The mobile platform has skid-steering / omnidirectional kinematics based on 4 high power motor wheels. Each wheel integrates a brushless motor with a high precision odometer sensor. The robot base can navigate autonomously or teleoperated by means of a PTZ camera that transmits video in real time. This camera is not used as we have an external camera. For teleoperating the mobile base we use human input. SUMMIT-XL STEEL can be configured with a wide range of sensors. It also has internal (USB, RS232, GPIO and RJ45) and external connectivity (USB, RJ45, power supplies 5, 12 VDC and battery) to add custom components easily such as the Franka Emika Panda arm. SUMMIT-XL STEEL uses the ROS open architecture.

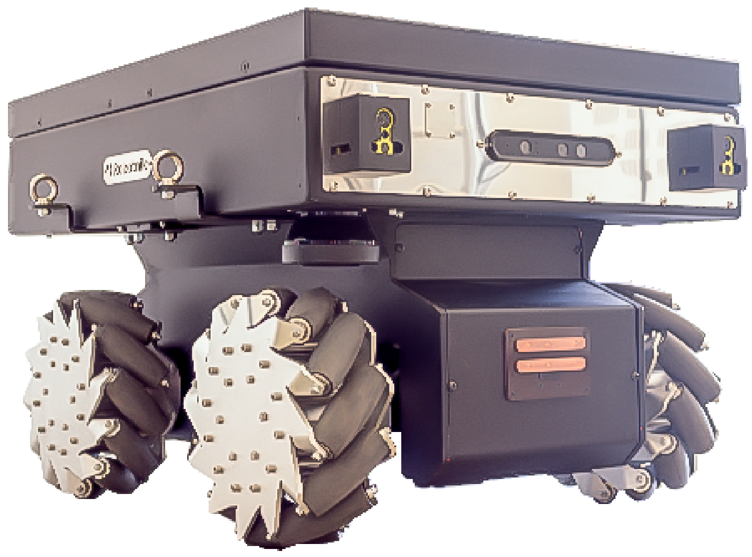


Figure 4.3: Robotnik Summit XL Steel

4.1.3 Intel RealSense D435i

The Intel RealSense D435i is a RGB-Depth camera. The camera uses stereo depth imaging for accurate depth maps. The resolution used is 640 X 480 at 30 frames per second (fps). This resolution was chosen as higher resolutions were not processed quickly enough by the system to be suitable for real-time execution.

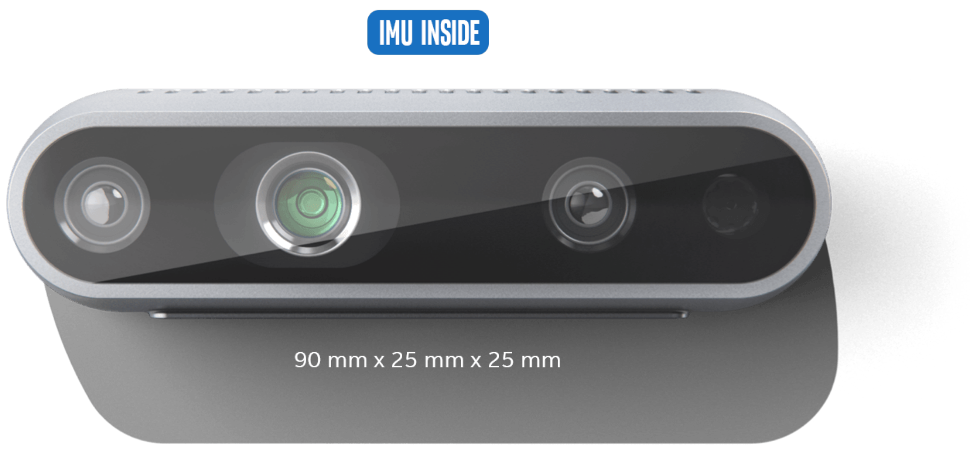


Figure 4.4: Intel RealSense D435i

Stereo Depth Vision

Stereo depth cameras project infrared light onto a scene to improve the accuracy of the data, but unlike coded or structured light cameras, stereo depth cameras can use any light to measure depth. For a stereo depth camera, all infrared noise is good noise. Stereo depth cameras have two sensors, spaced a small distance apart. A stereo depth camera takes the two images from these two sensors and compares them. Since the distance between the sensors is known, these comparisons give depth information. Stereo cameras work in a similar way to how we use two eyes for depth perception. Our brains calculate the difference between each eye. Objects closer to appear to move significantly from

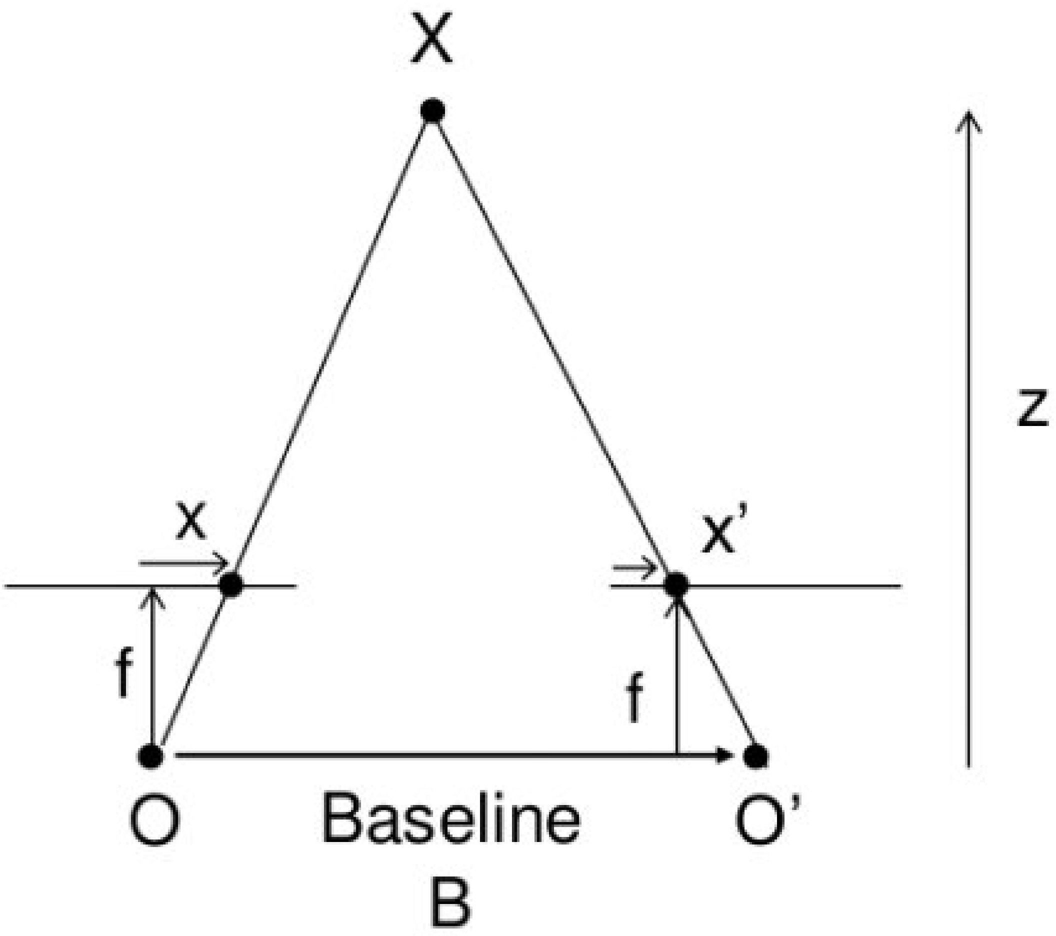


Figure 4.5: Stereo Depth Camera Working Principal [34]

eye to eye (or sensor to sensor), where an object in the far distance appear to move very little.

The Figure 4.5 contains equivalent triangles. Writing their equivalent equations will yield us result equation 4.1.

$$disparity = x - x' = (Bf)/Z \tag{4.1}$$

where x and x' are the distance between points in image plane corresponding to the scene point 3D and their camera center, B is the distance between two cameras (which we know) and f is the focal length of camera (already known). Z is the depth of the object. So in short, the above equation says that the depth of a point in a scene is inversely proportional to the difference in distance of corresponding image points and their camera centers. So with this information, we can derive the depth of all pixels in an image.

Because stereo depth cameras use any light to measure depth, they will work well in most lighting conditions including outdoors. The addition of an infrared projector means that in low lighting conditions, the camera can still perceive depth details.

4.2 Software Tools

4.2.1 YOLO (You Only Look Once)

YOLO is a singular Convolutional Neural Network designed for fast object recognition purposes. It takes RGB images as input and outputs the bounding box with its probability. A popular neural network which provides such an output is a Region Based Convolutional Neural Network (R-CNN). In a R-CNN, a region proposal method is used to generate the output. It uses a part of the image, known as a region to train the model to detect objects. A bounding box is first produced and a classifier is then applied to it to generate the probabilities for it. Following classification, the bounding boxes are further refined, duplicates are eliminated and are re-scored based on other objects in the image. Unsurprisingly, this approach is very slow to train and test making it less optimal in real-time applications. YOLO solves this approach by incorporating all these processes in a single CNN. It simplifies the entire pipeline from the input, image pixels to output, bounding box with probabilities making it significantly faster than the R-CNN. This also means that since there are no stages, the input image has to be looked at only once, hence the name You Only Look Once (YOLO). YOLO treats the classification as a regression problem. The main benefits of YOLO over other approaches are:

- Fast Performance at 45 fps with a latency of less than 25 ms.
- Lower background error rates than R-CNN as it uses the entire image to train and infer.
- Accuracy is comparable to state of the art object detection systems.
- Can be used in varying situations as it is trained on the images with contextual information.

How YOLO Works?

YOLO divides the input image into an S × S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. If no object exists in that cell, the confidence scores should be zero. Otherwise we want the confidence score to equal the Intersection Over Union (IOU) between the predicted box and the ground truth. IOU is defined as the the ratio of area of overlap and area of union of the bounding boxes as shown in Figure 4.6.

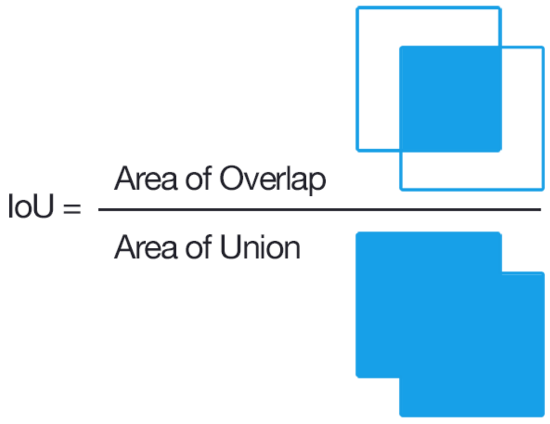


Figure 4.6: Intersection over Union [39]

Each bounding box contains 5 predictions, 4 coordinates of the bounding box (*xmax*, *xmin*, *ymax*, *ymin*) and confidence which is the IOU value for the box with respect to the ground truth. Figure 4.7 shows these steps.

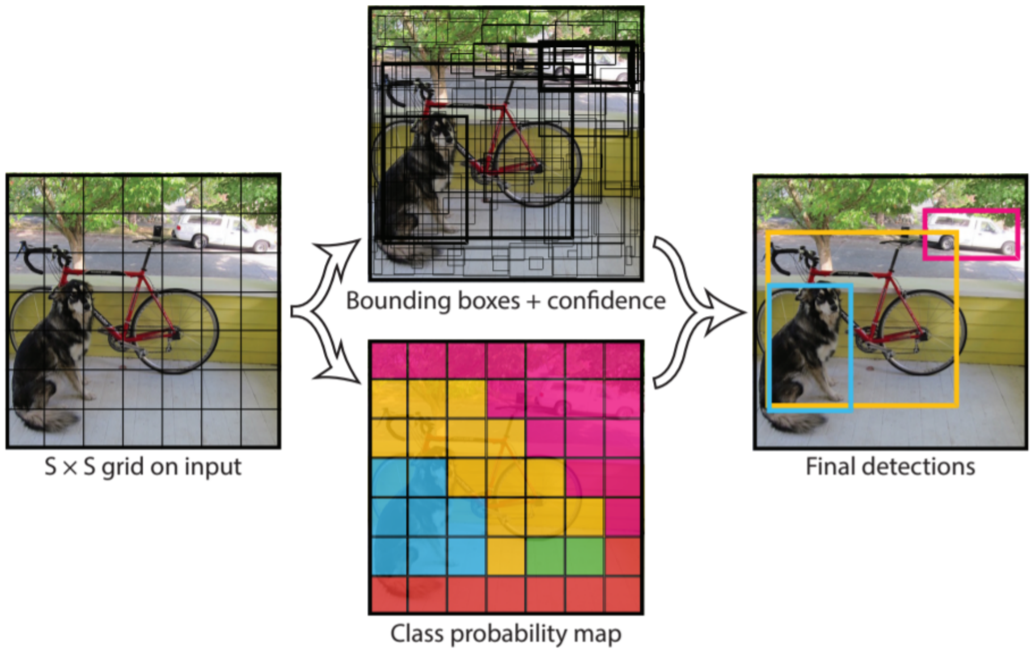


Figure 4.7: YOLO Object Detection Methodology [37]

Structure of the CNN

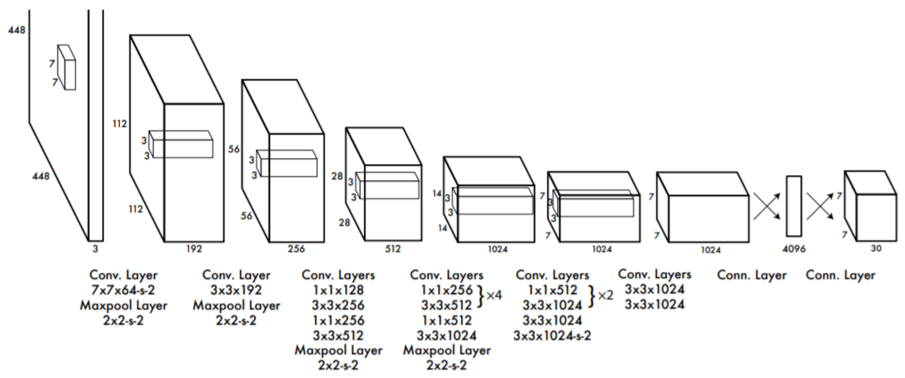
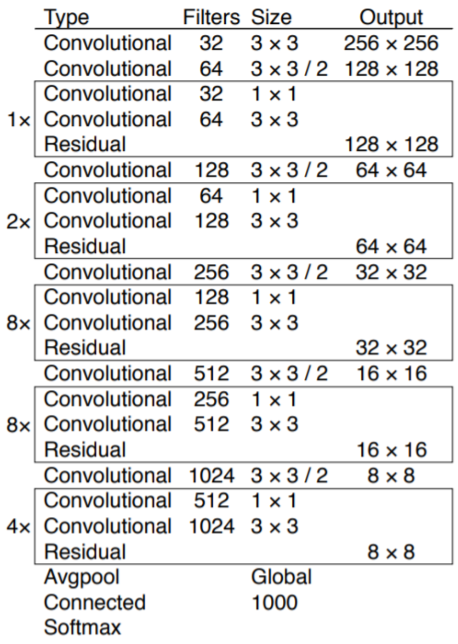


Figure 4.8: Structure of YOLO [37]

YOLO consists of 24 convolutional layers followed by 2 fully interconnected layers as shown in Figure 4.8. This structure was updated to 53 convolutional layers in YOLOv3 as shown in Table 4.1 which is what is used in the framework proposed in this thesis.



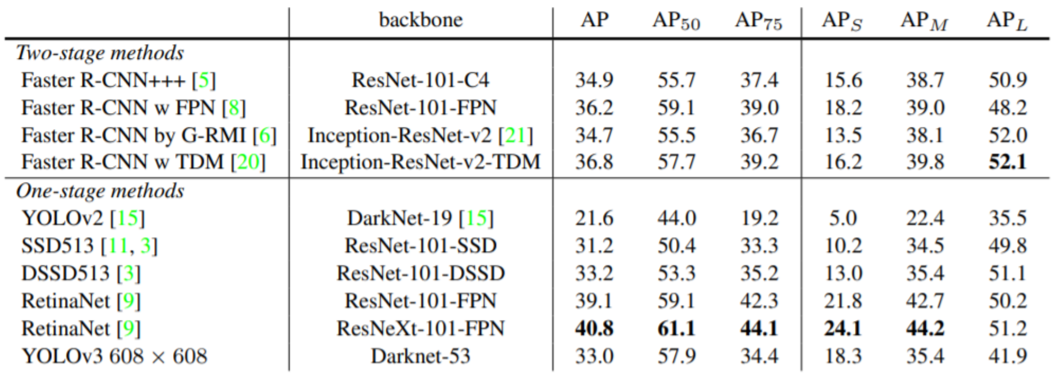
	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Table 4.1: YOLOv3 Structure [38]

YOLOv3 improves on YOLO in the following aspects:

- Improved localized detection to detect smaller objects. It was achieved by normalizing the training dataset making the layers learn more evenly and independently.
- Applying k-means clustering to training images to give priors to the network for choosing bounding boxes.
- Introducing multiple size bounding boxes for each object to find optimal IOU.
- Using linear regression to predict objectness score. Objectness score tells us how well the bounding box fits the object with respect to the ground truth.

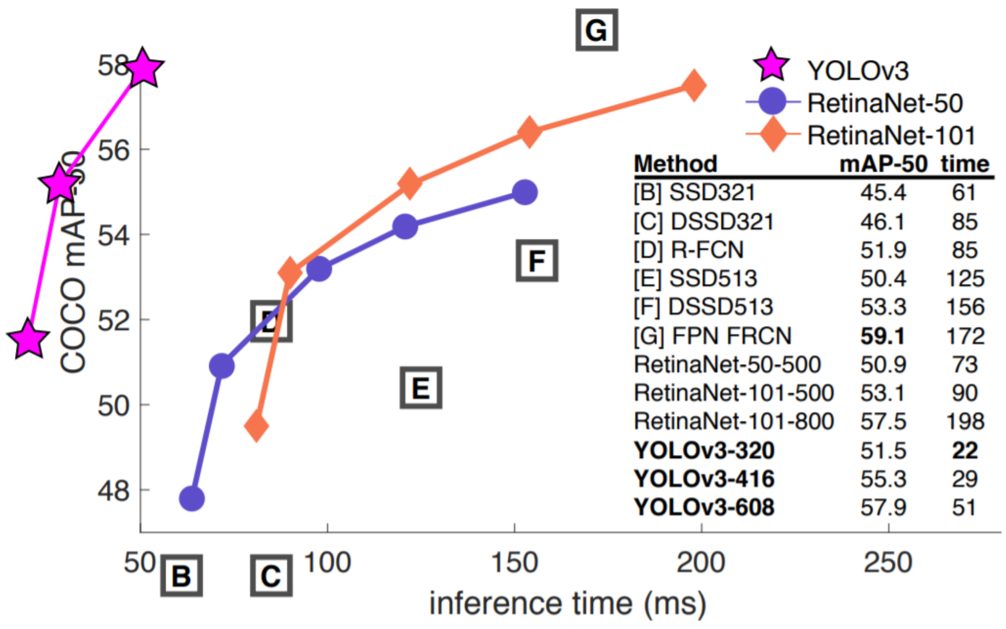
All these improvements result in almost twice the performance than the original version of YOLO. We can see how YOLOv3 compares with other object detection systems in Table 4.2.



	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [5]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [8]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [6]	Inception-ResNet-v2 [21]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [20]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [15]	DarkNet-19 [15]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [11, 3]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [3]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [9]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [9]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608×608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Table 4.2: Performance of YOLOv3 compared with other object detection systems. [38]

YOLOv3 performs at par with other systems when compared in the AP_{50} metric. It suffers with smaller objects as compared to others, but there is a significant improvement than YOLOv2 which itself improves on YOLO. The main thing to note here is that, YOLO is as much as 3.8× faster than RetinaNet which



Method	**mAP-50**	**time**
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

Figure 4.9: Speed vs Accuracy tradeoff at AP_{50} [38]

is the state of the art approach and yet it comes close to matching its accuracy. A comparison of YOLOv3 with other systems with respect to the speed/AP ratio is seen in Figure 4.9.

Inference time is the time required to process the output. We can see that YOLOv3 performs almost four times faster than RetinaNet despite maintaining high enough accuracy. The unit used to measure accuracy is the mAP AP_{50} measure from the COCO dataset [30]. COCO is a large-scale object detection, segmentation, and captioning dataset developed by Microsoft.

4.2.2 PyQt5

PyQT5 is a Graphical User Interface (GUI) framework for Python. It is very popular among developers and the GUI can be created by coding or a QT designer. A QT Development framework is a visual framework that allows drag and drop of widgets to build user interfaces. PyQt is a mature set of Python bindings to Qt for cross-platform development of desktop apps. It offers a rich selection of built-in widgets and tools for custom widgets creation to shape sophisticated GUIs.

4.3 Proposed End-User Framework for Robot Control

In this section the proposed end-user framework for robot control is presented.

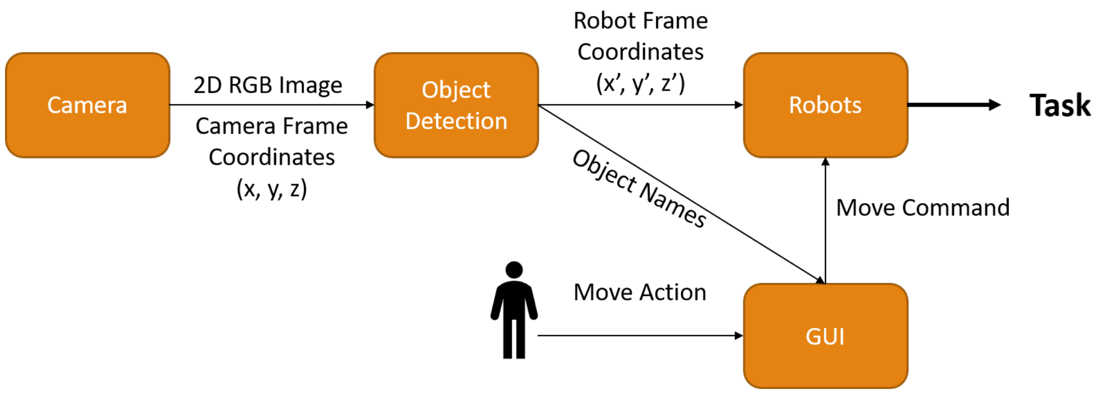


Figure 4.10: System Architecture Diagram

Figure 4.10 shows how the various components in the system interact with each other to perform any instructed task from the user. The entire system is built using two ROS Nodes. The first contains the Camera Publisher, YOLO subscriber, ROS TF Listener and move group interface and the other for YOLO Publisher and Camera subscriber. There are additional ROS tf nodes which publish the static transforms for the camera, summit and panda robots. The camera, YOLO, tf and move group were integrated in a single node to reduce communication delays caused by message passing.

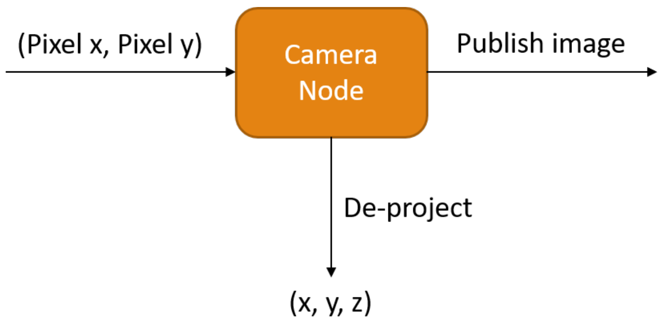


Figure 4.11: Camera Node

4.3.1 Camera Module

The camera module interfaces the Intel RealSense camera with the ROS framework. We use the pyrealsense2 library to get the image frames from the camera which is connected to the computer with a USB 2.0 connection. The image frame needs to be converted to an image message to be published on a ROS Topic of the camera (/camera/image/image_raw). We use the cv_bridge library for this operation. An image frame is published at the frame rate (30 fps) specified from the code creating a video stream. pyrealsense2 is also used to get the real world coordinates of the object to be picked by passing its pixel position from the video stream. This process is called as de-projection. This process uses the depth sensor present in the camera.

4.3.1.1 Camera Publisher

The camera publisher sends an image message on the ROS topic /camera/image/image_raw. The working of this is explained in Figure 4.12.

The color image frame is retrieved from the camera sensor using pyrealsense2. NumPy [20] converts the color image frame to a OpenCV [24] im-

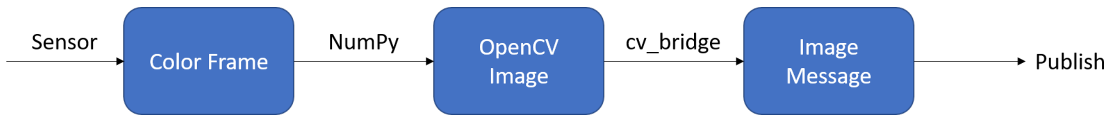


Figure 4.12: Camera Publisher

age which is a numpy array. This image is converted to a Image.msg using cv_bridge library and is published on /camera/image/image_raw ROS topic. This process repeats every time a new image frame is received from the camera sensor at a fixed frame rate which is 30 fps.

4.3.2 Object Detection Module

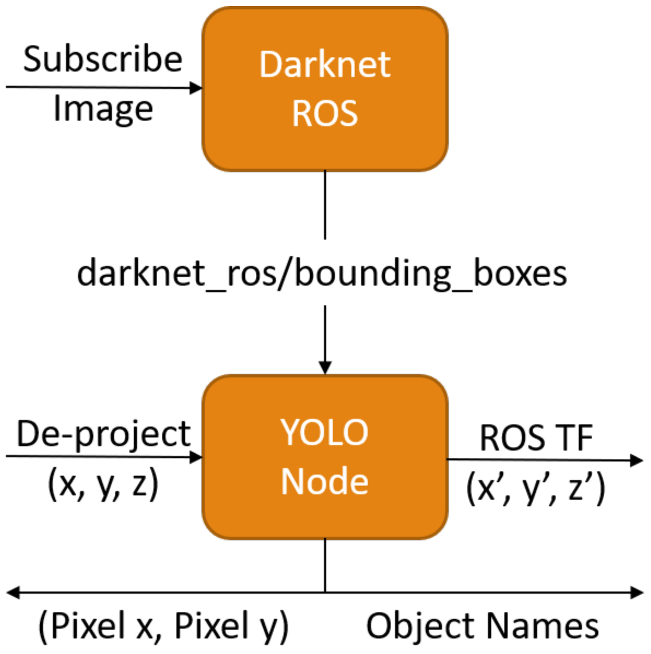


Figure 4.13: Object Detection Module

The object detection module has to perform object detection using YOLO and get the real-world coordinates for the object in the panda frame of reference. YOLO requires a video stream as input and it publishes the bounding box information as a bounding box message on a ROS topic (darknet_ros/bounding_boxes). The bounding boxes can be seen in Figure 4.14 as boxes surrounding recognized objects.

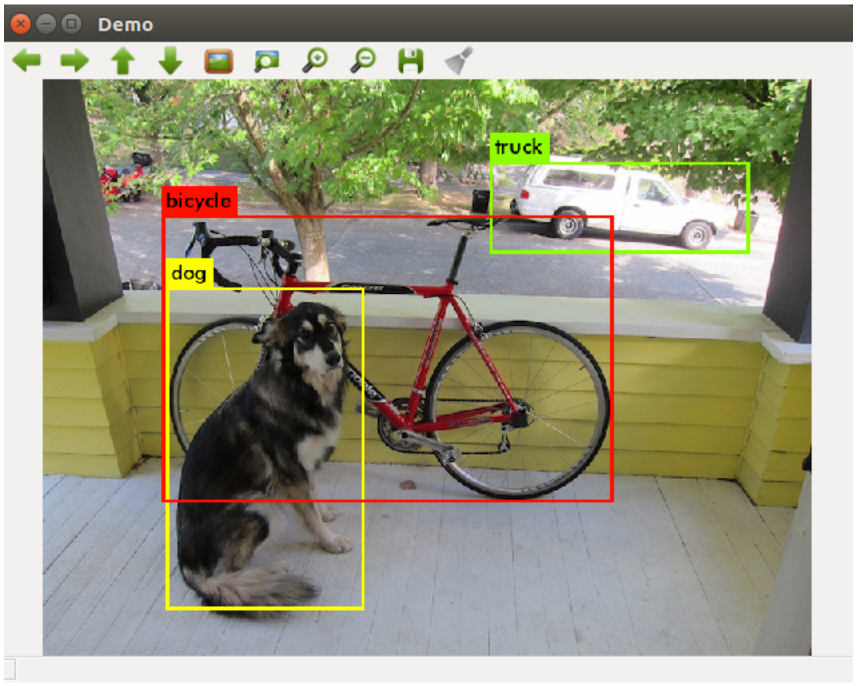


Figure 4.14: YOLO Output

A bounding box message contains the name of the object the box surrounds and xmax, xmin, ymax and ymin values of the box. Assuming the object is in the center of the box we can find its pixel position by calculating the centroid of the box by using equations 4.2.

$$x_p = (xmax + xmin)/2 \tag{4.2}$$

$$y_p = (ymax + ymin)/2 \tag{4.3}$$

x_p and y_p are used to obtain real-world (x, y, z) coordinates using de-projection from the camera module. Bounding boxes are published every time a new image frame is received from the camera. The process of centroid calculation takes place every time a bounding box is published. We can use this to track objects in real-time by dynamically updating their locations every time a new bounding box for the object is published. The (x, y, z) obtained from deprojection are in the camera_color_optical_frame and need to converted to the panda's frame which is world. To convert we can publish the coordinates to ROS tf and listen to the transformed coordinates. This process takes place when we want to pick a selected object from the GUI.

4.3.2.1 Camera Subscriber and YOLO Publisher

This pair of subscriber and publisher is part of the darknet_ros node. This node runs the YOLO neural network. For the CNN to have the input the camera subscriber is used to get the video feed from the /camera/image/image_raw ROS Topic. The output of YOLO is the bounding box. The YOLO Publisher combines all the bounding boxes for each image message and publishes this array of bounding_box's on the /darknet_ros/bounding_boxes ROS Topic.

4.3.2.2 YOLO Subscriber

YOLO publishes the bounding box information on a ROS topic /darknet_ros/bounding_boxes. It publishes a bounding boxes msg which is an array of bounding boxes. The YOLO subscriber aceepts this array of bounding boxes and parses through it. If it finds an object name it has previously stored, it updates its pixel position. If a new object is found it will create a new entry storing the object name and its pixel position.

4.3.2.3 ROS TF Listener

A ROS TF Listener is a subscriber to the ROS TF Publisher. The publisher node is initialized separately from the main node. Unlike a traditional subscriber, a ROS TF listener does not have a callback function. The publisher keeps on updating the transforms in the background continuously. The ROS TF listener will when required to transform the point will send the target frame, the source frame and the point in the source frame. The listener will return the transformed coordinate in the target frame.

4.3.3 Robot Module

The robot module performs all the robot movements. This robot is controlled by specifying velocities, joint angles or Cartesian coordinates in the panda's frame of reference (world). MoveIt is used as a motion planning framework. Figure 4.15 shows the structure of the MoveIt Framework. MoveIt provides a move_group interface which is implemented using a moveit_commander namespace. This namespace provides us with a MoveGroupCommander class, a PlanningSceneInterface class, and a RobotCommander class. All the movements are performed using objects of the MoveGroupCommander class using Cartesian coordinates and joint angles for the home pose. The PlanningSceneInterface class is used to add the camera mounting hardware in the environment to avoid collision with it using the built in collision detection provided by the move group interface. Move group applies inverse kinematics using the coordinates to get the desired joint angles for the robot to move to the specified position. It uses popular motion planning libraries such as OMPL (Open Mo-

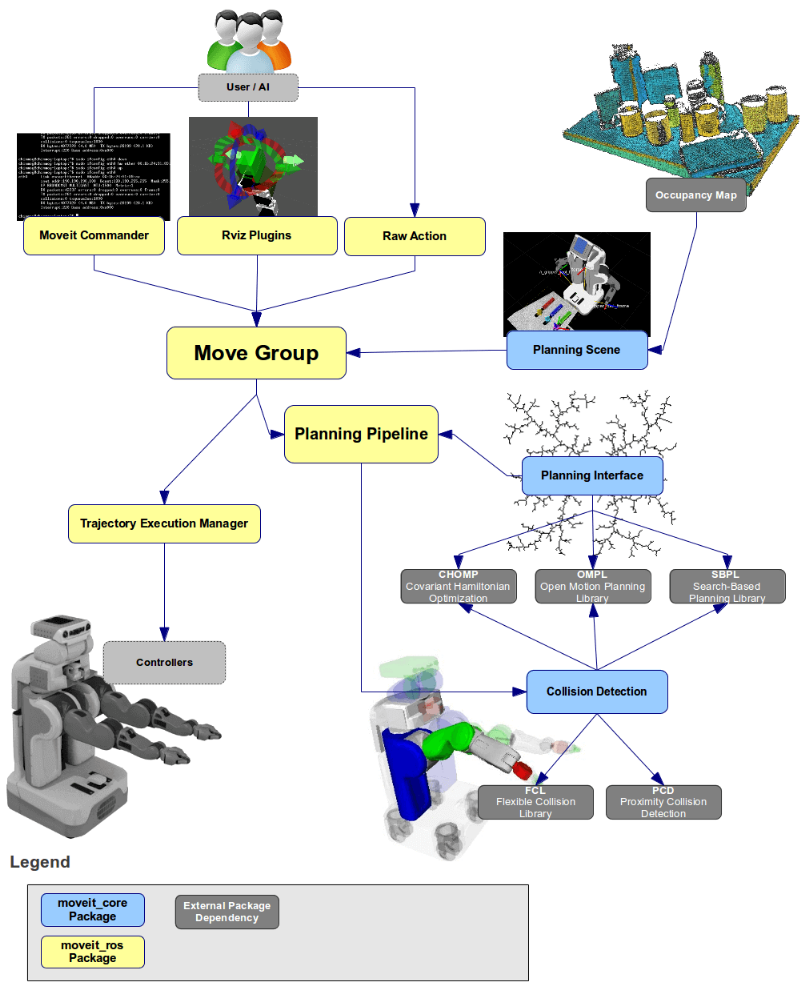


Figure 4.15: Robot Control Framework [33]

tion Planning Library) to find optimal values for the joint angles. The action of the robot is performed by the Trajectory Execution Manager using a JointTrajectoryAction which is connected to the franka_ros robot controller. To move the summit it requires the velocities to be published on a ROS Topic which the controller is subscribed to. The velocities are published as a twist message on the ROS topic.

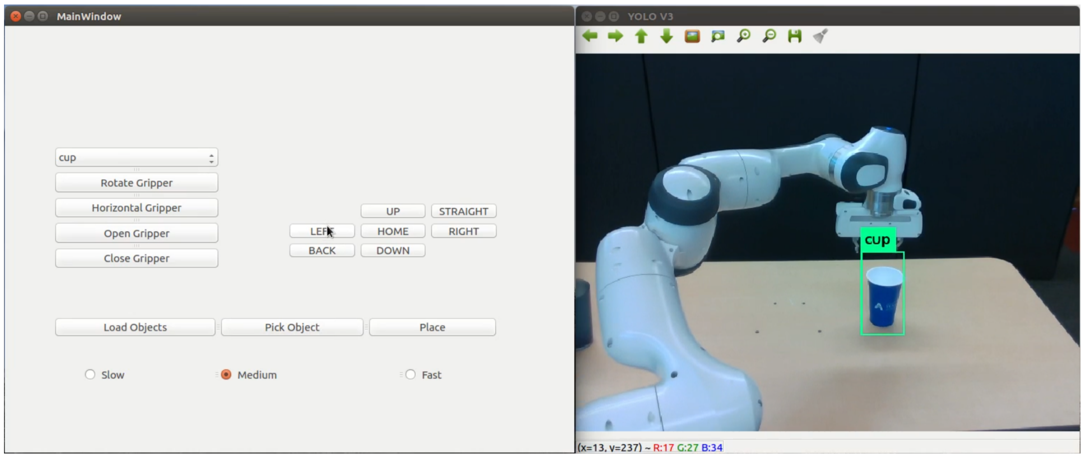


Figure 4.16: Graphical User Interface with YOLO Output

4.3.4 The Graphical User Interface Module

The Graphical User Interface (GUI) is built using the PyQt5 library. Any python library can be used to make the GUI as long as it implements a class which contains all the user interface elements, such as buttons, labels etc and can process keyboard input. The primary goals of the GUI were to making the robotic system easy to move, easily select objects for pick and place and design safety features to avoid collisions of the robots with themselves and with objects in the environment The most simplest of controls for the mobile base is using the keyboard for movement using the W, A, S, D keys. The summit moves in a fixed velocity in the desired direction. Rotation of the robot is implemented by pressing the W, A together for anti-clockwise rotation and pressing W, D keys together for clockwise rotation. To stop the mobile base from moving one simply has to stop pressing the control keys. Since the mobile base does not move at a high enough speed, implementing braking separately was not necessary. For the robotic arm, having simple step based motion based controls was chosen to be the safest option as the user cannot accidentally induce jerks in the robotic arm causing it to degrade or even break in the worst case. Various end-effector orientations were implemented to give the user the choice to grip the object from the top, side or front. The default is vertical orientation which can be used pick up an object from the top. Rotate Gripper orientation can be used to pick up the object from the side and horizontal orientation can be used to grip an object from the front. Motion control of the arm is implemented using a combination of directional buttons and radio buttons for step size. The slow option takes small steps of 1 cm, medium takes steps of 3 cm and fast takes steps of length 5 cm. The user can see the YOLO output beside the GUI so he can chose to pick up objects identified in the scene automatically using the pick button which moves the gripper to the front/top of the object depending on the gripper orientation. The user can then move to the robot manually to grasp the object from any part. The place button places the object at a pre-defined location specified in the code. Objects detected by YOLO are loaded into the combo box for selection using the load objects button. Object locations are dynamically updated, therefore the user does not need to use this button unless a new object is added into the scene or the combo box is empty.

4.3.5 Computer Setup

This section explains the details of the computer used to run the application.

4.3.5.1 Computer Specifications

Table 4.17 provides the information about the Computer Specifications

Processor	Intel Core i5-2400
GPU	NVIDIA GTX 1050Ti (4 GB VRAM)
GPU Driver	411.xx
RAM	16 GB
OS	Ubuntu 16.04 LTS
ROS Version	Kinetic
Linux Kernel	Liquorix 5.8

Figure 4.17: Computer Specifications

4.3.5.2 Linux Kernel

Franka Emika, the makers of the panda robot recommend using a Real Time (RT) linux kernel for the machine used to connect to the robot arm. Using a RT Kernel is not possible in our case as there is also a NVIDIA GPU installed in our system to run YOLO. The NVIDIA GPU does not have drivers for a RT Kernel. If we decide to run YOLO just on the CPU, the system would not run in real-time as performance is very poor (2-3 fps). The solution for this is to use a PREEMPT-RT Kernel for which NVIDIA Drivers are available. On its own if an PREEMPT-RT Kernel is used, the code would not run as the franka controllers would throw errors for the missing RT Kernel. To solve this issue, the franka library libfranka was patched to remove all RT Kernel checks which threw the error. The kernel was also patched to include the RT Flags making the controller think the system is running on an RT Kernel. Though this is not the recommended software setup for the computer, in our testing using the patched kernel and libfranka was not an issue. The robotic system ran reliably and all the safety mechanisms included with the franka controllers were found to work as intended.

4.3.5.3 Docker Container

For the system to be used on another machines, the user cannot be expected to have all the libraries and drivers setup on his machine. For this reason a docker container is being implemented and will be uploaded to docker hub with the host requiring only to run the patched kernel which is uploaded to git along with the setup instructions for it.
Link to the patched kernel: - https://github.com/heracleia/lab_kernel.git

CHAPTER 5

EVALUATION

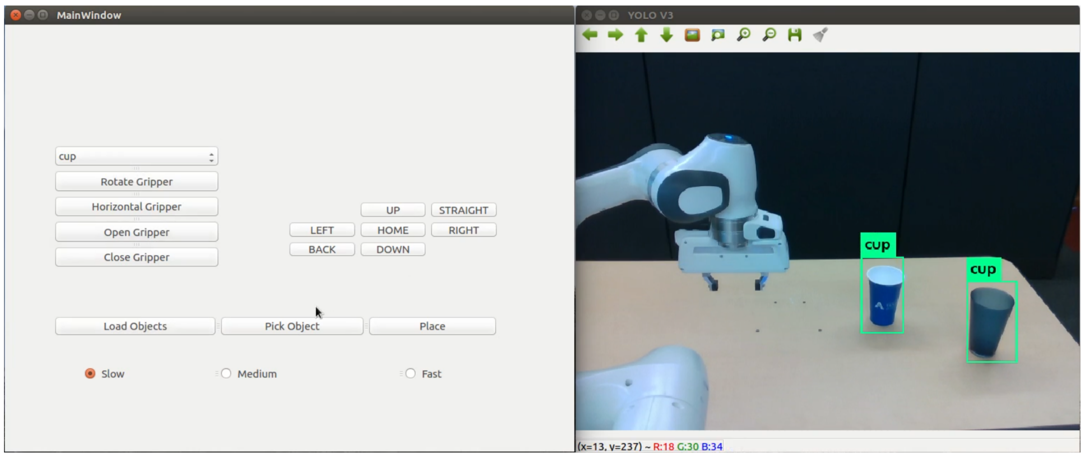
To evaluate the system, we can test the working in a practical use-case scenario such as pick and place tasks in various modes of operation. Apart from the scenarios tested, the potential use of the robotic system is described to be used in a hospital environment as a nurse assistant. To test the usability of the system, the methodology proposed by Weiss et al. [50] is used.

5.1 Scenarios

There are two scenarios discussed which are dependant on the object being detected or not. Since there is not a physical button to switch between the semi-automated and teleoperation mode, it is decided by the actions the user performs using the GUI. If the user is able to load objects from the scene (the combo box for object selection has an object upon pressing the load object button), he/she can use the pick object button to move the robot arm to the vicinity of the object and grasp the object manually. This mode of operation is defined as the semi-automated mode. It is called as semi-automated and not fully automated as the user has to manually grasp the object. If the user chooses to move the robot arm manually using the directional buttons to the object it is defined as operating using the manual mode or teleoperation. The teleoperation mode is needed as YOLO is only able to detect objects if it is close enough to the camera. YOLO was also found to be unreliable in identifying common objects in sub-optimal lighting. Having the ability to control the robotic system manually solves these issues as the robot can be moved closer using the mobile base for the objects to be detected reliably. In the worst case that object is still not detected, the user can manually move the robot to complete the task.

5.1.1 Scenario 1: Object Stacking in Semi-Automated Mode

In this scenario several cups are placed on a table in front of the robot. The task is to stack the cups together. For this operation, the user uses the semi-automated mode of operation. In this mode, the user presses the pick object button to move the robot arm to the object. The orientation of the gripper is at default, therefore the robotic arm moves to the top of the cup as seen in Figure 5.2a. The user manually moves the robot arm using the buttons to pick up the cup from the side as seen in Figure 5.2b. The user can then press the place button to place the cup at a pre-defined coordinate seen in Figure 5.2c. The user repeats the procedure for the second cup making sure to grasp it from the side and placing using the place button to stack the two cups as seen in Figure 5.2d.

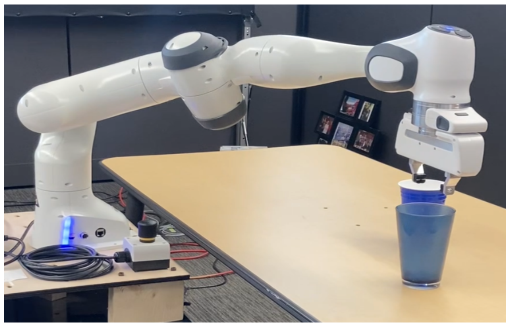


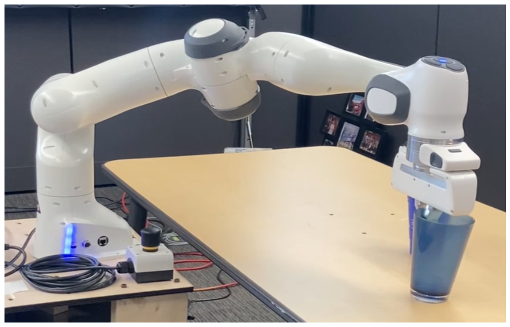
cup

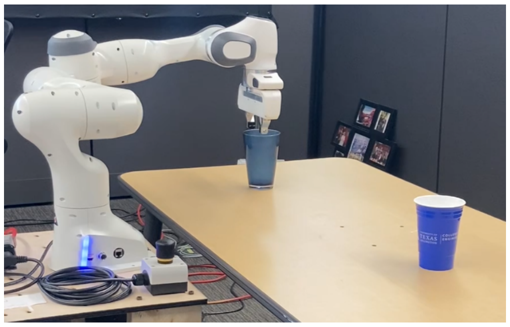
[✓ Slow] Medium Fast

(x=13, y=237) ~ R:18 G:30 B:34

Figure 5.1: Scenario with two cups


(a) Pre-Grasp pose


(b) Grasp pose


(c) Place pose

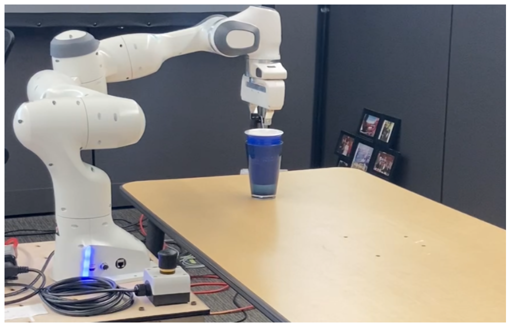

(d) Task Complete

Figure 5.2: Semi-Automated Scenario: Object Stacking

5.1.2 Scenario 2: Pick and Place Task in Manual Operation (Teleoperation)

In this scenario, the object to be picked and placed is not detected by YOLO. The user moves the arm manually using the GUI to pick and place an object. Since the object is not detected, the load objects button and the pick object button are not active. The place button is not dependent on the object, therefore we can still use it to place the object at a certain pre-defined position. Figure 5.3 is one such case where teleoperation is needed. Figure 5.4 showcases the object being grasped from the side using the rotate gripper orientation. It can be placed using the place button are wherever the user chooses by manually moving the robot arm, mobile base or both.

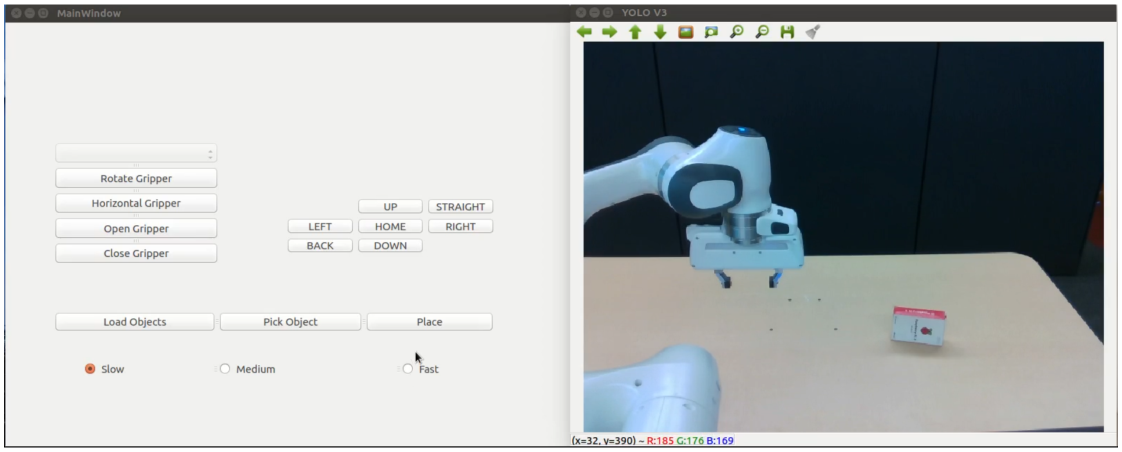


Figure 5.3: Teleoperation GUI

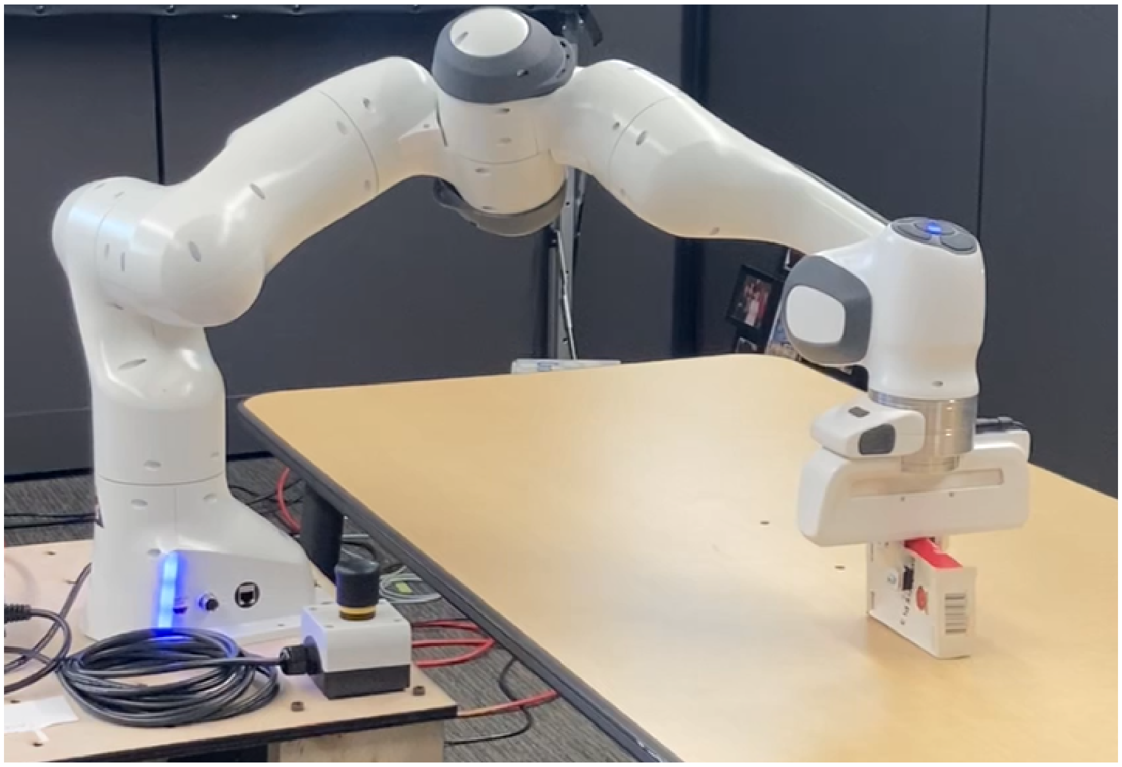


Figure 5.4: Teleoperation in action

5.1.3 Mobile Intelligent Nurse Assistant

The role of robotics in healthcare has been growing in the last decade [29]. This is due to the need to improve the quality and safety of care while controlling expenses [12]. Moreover, the shortage of nurses and healthcare personnel significantly impacts the quality of care [18, 46]. The COVID-19 pandemic demonstrated how vulnerable the healthcare workers are and at risk to be exposed to the virus [5].

Several robotic systems are introduced to assist with hospital logistics, disinfection of spaces, and patient screening [25, 4]. Robots have the potential to support caregivers in several routine tasks, while caregivers can focus on the actual patient care. For example, there is a variety of commercially available robots that are currently used in hospitals to take care of delivery tasks, such as the mobile robots TUG [49] and Relay [48], or fetching objects, such as Moxi [40], a mobile robot equipped with a robotic arm

The robotic system implemented in this thesis can be used by the nurses to fetch objects for the patient, use as a walking aid for patients using the robotic arm as support, disinfect spaces or operate machinery by using the robot arm to press buttons of a machine using teleoperation.

5.2 Proposed Evaluation

Due to the ongoing COVID-19 pandemic, a comprehensive user study was not possible to be conducted. The methodology used to evaluate user data is detailed in this thesis. According to [50], to test the usability of a HRI system, the following factors must be evaluated:

- Effectiveness
- Efficiency

- Learnability
- Flexibility
- Robustness
- Utility

The next sections provide information on how these factors are evaluated.

5.2.1 Effectiveness Evaluation

The effectiveness measure aims to quantify how well the system works in successfully completing a given task. It can be measured by the ″success rate″ or ″task completion rate″. To obtain this data the user is asked to complete the following tasks:

- Pick and Place a specific object from a table reachable by the arm.
- Use teleoperation to fetch any object from the environment.
- Use semi-automated mode to fetch any object from the environment.

The user is asked to perform each task 10 times, and the number of attempts required to complete each task is recorded. The goal is to have at an average score of 1.1 for all the tasks. This score for each task is calculated by equation 5.1. An attempt is defined as a sequence of operations done by the user to perform the task successfully without having to abort back to the start state either by pressing the home button or restarting the robotic system.

$$score = totalno.of attempts/10 \tag{5.1}$$

If the user attempts to perform the task 11 times meaning he/she will fails to complete the task in one attempt but successfully completes in another attempt the total attempts will be 9 successful attempts + 1 failed attempt + 1 successful attempt for the failed attempt = 11 attempts. Using equation 5.1 we get the score as 1.1 for a particular task.

5.2.2 Efficiency Evaluation

The efficiency measure aims to measure the speed of operation of the robot for a task. This measure is not particularly important for our system as the goal is not to decrease time required for a specific task but to enable the user to do a wide variety of tasks remotely. That being said, the system should perform the task fast enough so that the user prefers it over doing it himself. This can be evaluated by asking the user to perform the tasks mentioned in the previous section and then asking them to score the efficiency on a likert scale from 0-10 with 0 being too slow and 10 being too fast. The mode of the values gives us the overall user efficiency of the system. Efficiency of the system can be further tested by conducting the tests when running the software on different machines or the user's personal machine.

5.2.3 Learnability Evaluation

As the name suggests, learnability defines how easy is the system to use for a novice user. In this measure we want to gauge how familiar is the GUI to the user, are the actions performed be the controls consistent and predictable and how generalized is the system.

5.2.4 Familiarity

To measure familiarity, the user can be shown different user interfaces and asked to chose which interface they feel more comfortable using. The measure of the success of the GUI can be measured by the number of users that pick the GUI implemented in this thesis.

5.2.5 Consistency and Predictability

To measure consistency and predictability, the user can be asked to predict the action performed by the robot if pressed a certain button. The system can be then scored on the number of times the user is able to predict accurately. Consistency can also be measured additionally performing an action such as picking up and object repeatedly.

5.2.6 Robustness

Robustness is a measure of the effectiveness of the fail-safe systems implemented in the robotic system. This measure is an important aspect as the system is intended for use by novice users. Fail-safe mechanisms implemented in the system will help keep the system operational even if the user moves the robot in ways that can damage it. To evaluate the robustness of the system, a user study can be conducted where the user can be asked to use the robot in ways that it can damage itself. For e.g, the user can press the pick object without an object in the scene, try to jam the robot arm by moving it beyond its joint limits and intentionally moving the robot so it hits the camera pole behind it. The experiment can be evaluating if the user is successful in doing these tasks. The user can also be asked to find ways in which he/she thinks the software can be crashed.

5.2.7 Utility

Utility of the system is a combination of effectiveness and efficiency. This measure helps us understand how a given interface helps the user to achieve his/her goal or perform a specific task. Effectiveness was measured as success rate and efficiency was measured using likert scale data. To combine these two, the likert scale data is converted to percentage by 5.2. The effectiveness measure can also be converted to percentage using 5.3.

$$EfficiencyPercentage = (mode/10)x100 \tag{5.2}$$

$$Effectiveness Percentage = (100 - (score - 1) x 100) \tag{5.3}$$

The average of these 2 percentages gives us an utility score.

CHAPTER 6

CONCLUSION AND FUTURE WORK

The broad vision of this thesis was to implement an end-user framework for robot control which allowed novice users to perform common tasks such as pick and place with ease by teleoperation. The work presented in this thesis provides the user with the following abilities:

- Operating the robotic system which includes the robotic arm and mobile base using an intuitive and simple graphical user interface.
- Performing pick and place using a semi-automated mode or teleoperation.
- Selecting objects for pick and place through the GUI for semi-automated pick and place.

To develop the framework, the open-source Robot Operating System (ROS) was used along with the MoveIt framework to provide motion control for the robotic arm. The mobile base was controlled using the summit controller which was interfaced directly through ROS. Object detection used deep learning techniques to perform object detection in real-time. Computer vision techniques were utilized to transform camera images to ROS messages thus integrating the camera into the framework.

Chapter 1 discussed the motivation for the thesis and the objectives for the framework were outlined. In Chapter 2, the related work in building user-interface for human robot collaborative systems was detailed. Various approaches such as block based programming, task based programming and traditional code based programming approaches were analyzed. Drawbacks in these approaches were analyzed in these in building the framework.

Chapter 3 provided some background knowledge required for the understanding of the framework. The basics of ROS, MoveIt, RGB-Depth Cameras and Neural Networks was explained. Chapter 4 explained the proposed framework and explained all the components, both hardware and software. Details about the implemented ROS Publishers, Subscribers and their relations were explained on detail. The user interface implemented was discussed and the functionality of each element in the GUI was detailed.

Chapter 5 explained the evaluation methods used to understand the usability of the system. Some practical scenarios were explained from as how the user operates the system in a scenario. The two modes of operation semi-automated and teleoperation were discussed in regards to why and how an user would use these in a practical world scenario. An application of the system as a Medical Intelligent Nurse Assistant was explained to demonstrate the operational efficiency of the framework. The testing methodology for a user study was explained.

6.1 Future Work

Although the framework achieves all the objectives mentioned in this thesis, there are a few areas where the work from this thesis can be extended to build an even more comprehensive framework. The following areas were identified which could be improved:

- Improving the camera performance on limited bandwidth connections such as Ethernet.

- Automating the entire pick object pipeline by gripper orientation estimation.
- Improving the object detection system to produce unique bounding boxes for objects of the same class.
- Wireless interface for communication between robot and computer.
- Conducting the user evaluation once COVID-19 restrictions are lifted.

BIBLIOGRAPHY

[1] Gopika Ajaykumar and Chien-Ming Huang. User needs and design opportunities in end-user robot programming. In *Companion of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, pages 93–95, 2020.

[2] Hend Al Tair, Tarek Taha, Mahmoud Al-Qutayri, and Jorge Dias. Decentralized multi-agent pomdps framework for humans-robots teamwork coordination in search and rescue. In *2015 International Conference on Information and Communication Technology Research (ICTRC)*, pages 210–213. IEEE, 2015.

[3] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[4] Laura Aymerich-Franch and Iliana Ferrer. The implementation of social robots during the covid-19 pandemic. *arXiv preprint arXiv:2007.03941*, 2020.

[5] Zackary D Berger, Nicholas G Evans, Alexandra L Phelan, and Ross D Silverman. Covid-19: control measures must be equitable and inclusive, 2020.

[6] ZM Bi, Mingchao Luo, Zhonghua Miao, Bing Zhang, WJ Zhang, and Lihui Wang. Safety assurance mechanisms of collaborative robotic systems in manufacturing. *Robotics and Computer-Integrated Manufacturing*, 67:102022, 2021.

[7] Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. Survey: Robot programming by demonstration. Technical report, Springrer, 2008.

[8] Cynthia Breazeal. Social robots: from research to commercialization. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, pages 1–1, 2017.

[9] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. Reducing the barrier to entry of complex robotic software: a moveit! case study. *arXiv preprint arXiv:1404.3785*, 2014.

[10] Mirko Daniele Comparetti, Elisa Beretta, Mirko Kunze, Elena De Momi, Jörg Raczkowsky, and Giancarlo Ferrigno. Event-based device-behavior

switching in surgical human-robot interaction. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 1877–1882. IEEE, 2014.

[11] Tom Coupe. Automation, job characteristics and job insecurity. *International Journal of Manpower*, 2019.

[12] Kathrin Cresswell, Sarah Cunningham-Burley, and Aziz Sheikh. Health care robotics: qualitative exploration of key challenges and future directions. *Journal of medical Internet research*, 20(7):e10410, 2018.

[13] E.Guizzo. Franka: A robot arm that's safe, low cost, and can replicate itself, 2016.

[14] Shirine El Zaatari, Mohamed Marei, Weidong Li, and Zahid Usman. Cobot programming for collaborative industrial tasks: An overview. *Robotics and Autonomous Systems*, 116:162–180, 2019.

[15] Claudia González, J Ernesto Solanes, Adolfo Muñoz, Luis Gracia, Vicent Girbés-Juan, and Josep Tornero. Advanced teleoperation and control system for industrial robots based on augmented virtuality and haptic feedback. *Journal of Manufacturing Systems*, 59:283–298, 2021.

[16] Axel Graser, Torsten Heyer, Leila Fotoohi, Uwe Lange, Henning Kampe, Bashar Enjarini, Stefan Heyer, Christos Fragkopoulos, and Danijela Ristic-Durrant. A supportive friend at work: Robotic workplace assistance for the disabled. *IEEE Robotics & Automation Magazine*, 20(4):148–159, 2013.

[17] Jérémie Guiochet, Mathilde Machin, and Hélène Waeselynck. Safety-critical advanced robots: A survey. *Robotics and Autonomous Systems*, 94:43–52, 2017.

[18] Jérémie Guiochet, Mathilde Machin, and Hélène Waeselynck. Safety-critical advanced robots: A survey. *Robotics and Autonomous Systems*, 94:43–52, 2017.

[19] Jérémie Guiochet, Mathilde Machin, and Hélène Waeselynck. Safety-critical advanced robots: A survey. *Robotics and Autonomous Systems*, 94:43–52, 2017.

[20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer,

Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[21] Intel. Beginner's guide to depth (updated). `https://www.intelrealsense.com/beginners-guide-to-depth/`.

[22] Tudor Ionescu, Sebastian Schlund, and Christina Schmidbauer. Epistemic debt: A concept and measure of technical ignorance in smart manufacturing, 02 2019.

[23] Tudor B Ionescu. Leveraging graphical user interface automation for generic robot programming. *Robotics*, 10(1):3, 2021.

[24] Itseez. Open source computer vision library. `https://github.com/itseez/opencv`, 2015.

[25] M Shamim Kaiser, Shamim Al Mamun, Mufti Mahmud, and Marzia Hoque Tania. Healthcare robots to combat covid-19. In *COVID-19: Prediction, Decision-Making, and Its Impacts*, pages 83–97. Springer, 2021.

[26] Maria Kyrarini. *Robot Learning from Human Demonstrations for Human-Robot Synergy*. PhD thesis, Universität Bremen, 2019.

[27] Maria Kyrarini, Muhammad Abdul Haseeb, Danijela Ristić-Durrant, and Axel Gräser. Robot learning of industrial assembly task via human demonstrations. *Autonomous Robots*, 43(1):239–257, 2019.

[28] Maria Kyrarini, Fotios Lygerakis, Akilesh Rajavenkatanarayanan, Christos Sevastopoulos, Harish Ram Nambiappan, Kodur Krishna Chaitanya, Ashwin Ramesh Babu, Joanne Mathew, and Fillia Makedon. A survey of robots in healthcare. *Technologies*, 9(1):8, 2021.

[29] Maria Kyrarini, Fotios Lygerakis, Akilesh Rajavenkatanarayanan, Christos Sevastopoulos, Harish Ram Nambiappan, Kodur Krishna Chaitanya, Ashwin Ramesh Babu, Joanne Mathew, and Fillia Makedon. A survey of robots in healthcare. *Technologies*, 9(1):8, 2021.

[30] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco:

Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[31] Marcus Mast, Michael Burmester, Birgit Graf, Florian Weisshardt, Georg Arbeiter, Michal Španěl, Zdeněk Materna, Pavel Smrž, and Gernot Kronreif. *Design of the Human-Robot Interaction for a Semi-Autonomous Service Robot to Assist Elderly People*, pages 15–29. Springer International Publishing, Cham, 2015.

[32] Abdul Montaqim. Offline programming software for industrial robots from robodk offers hundreds of virtual industrial robots from top robotics companies, 07 2015.

[33] MoveIt. Concepts. https://moveit.ros.org/documentation/concepts/.

[34] OpenCV Docs. Depth map from stereo images. https://www.docs.opencv.org/master/dd/d53/tutorial_py_depthmap.html.

[35] Lorenzo Peppoloni, Filippo Brizzi, Carlo Alberto Avizzano, and Emanuele Ruffaldi. Immersive ros-integrated framework for robot teleoperation. In *2015 IEEE Symposium on 3D User Interfaces (3DUI)*, pages 177–178, 2015.

[36] Alberto Poncela and Leticia Gallardo-Estrella. Command-based voice teleoperation of a mobile robot via a human-robot interface. *Robotica*, 33(1):1, 2015.

[37] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

[38] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[39] Ren Jie Tan. Breaking down mean average precision (map). https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52, 2019.

[40] Diligent Robotics. Moxi. https://www.diligentrobots.com/moxi/.

[41] Rui P Rocha, David Portugal, Micael Couceiro, Filipe Araújo, Paulo Menezes, and Jorge Lobo. The chopin project: Cooperation between hu-

man and robotic teams in catastrophic incidents. In *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–4. IEEE, 2013.

[42] Ujjwal Sharma Saumya Saxena. Introduction to deep learning. `https://www.geeksforgeeks.org/introduction-deep-learning/`.

[43] Max Schwarz, Jorg Stuckler, and Sven Behnke. Mobile teleoperation interfaces with adjustable autonomy for personal service robots. In *2014 9th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 288–289. IEEE, 2014.

[44] Sergey Dorodnicov. The basics of stereo depth vision. `https://www.intelrealsense.com/stereo-depth-vision-basics/`, 2018.

[45] Igor Sokolov, Alexander Misharin, Vasily Kupriyanovsky, Oleg Pokusaev, and Yulia Kupriyanovsky. Robots, autonomous robotic systems, artificial intelligence and the transformation of the market of transport and logistics services in the digitalization of the economy. *International Journal of Open Information Technologies*, 6(4):92–108, 2018.

[46] Igor Sokolov, Alexander Misharin, Vasily Kupriyanovsky, Oleg Pokusaev, and Yulia Kupriyanovsky. Robots, autonomous robotic systems, artificial intelligence and the transformation of the market of transport and logistics services in the digitalization of the economy. *International Journal of Open Information Technologies*, 6(4):92–108, 2018.

[47] Stanford Artificial Intelligence Laboratory et al. Robotic operating system.

[48] swisslog healthcare. Savioke relay® autonomous service robot for hospitals. `https://www.swisslog-healthcare.com/en-us/products-and-services/transport-automation/relay-autonomous-service-robot/`.

[49] TUG. Tug. `https://aethon.com/mobile-robots-for-healthcare`.

[50] Astrid Weiss, Regina Bernhaupt, Michael Lankes, and Manfred Tscheligi. The usus evaluation framework for human-robot interaction. *Proc. of AISB 09*, 4:11–26, 01 2009.

[51] Lijun Zhao, Yihuan Liu, Ke Wang, Peidong Liang, and Ruifeng Li. An

intuitive human robot interface for tele-operation. In *2016 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 454–459, 2016.

[52] Qijie Zhao, Dawei Tu, Shuo Xu, Hui Shao, and Qingxu Meng. Natural human-robot interaction for elderly and disabled healthcare application. In *2014 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 39–44. IEEE, 2014.