Vision-Based Control Using Object Detection and Depth Estimation for Robotic Pick and Place Tasks in Construction Applications

by

Sushilkumar Muralikumar

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

ARIZONA STATE UNIVERSITY

December 2022

ABSTRACT

The construction industry holds great promise for improvement through the use of robotic technologies in its workflow. Although this industry was an early adopter of such technologies, growth in construction robotics research and its integration into current construction projects is progressing slowly. Some significant factors that have contributed to the slow pace are high capital costs, low return on investments, and decreasing public infrastructure budgets. Consequently, there is a clear need to reduce the overall costs associated with new construction robotics technologies, which would enable greater dissemination.

One solution is to use a swarm robotics approach, in which a large group of relatively low-cost agents are employed to produce a target collective behavior. Given the development of deep learning algorithms for object detection and depth estimation, and novel technologies such as edge computing and augmented reality, it is becoming feasible to engineer low-cost swarm robotic systems that use a vision-only control approach. Toward this end, this thesis develops a vision-based controller for a mobile manipulator robot that relies only on visual feedback from a monocular camera and does not require prior information about the environment. The controller uses deep-learning based methods for object detection and depth estimation to accomplish material retrieval and deposition tasks. The controller is demonstrated in the Gazebo robot simulator for scenarios in which a mobile manipulator must autonomously identify, pick up, transport, and deposit individual blocks with specific colors and shapes. The thesis concludes with a discussion of possible future extensions to the proposed solution, including its scalability to swarm robotic systems.

ACKNOWLEDGMENTS

Words cannot express my gratitude to my advisor, Dr. Spring Berman, for allowing me to be part of the Autonomous and Collective Systems (ACS) lab and for her relentless patience and invaluable feedback over the years. She has been an incredible source of support and motivation, without which this thesis would not have come to fruition. I would also like to express my gratitude to Dr. Hamid Marvi and Dr. Sze Zheng Yong for their incredible classes on Robotics and Control and their constant support over the years.

I am incredibly grateful to members of the ACS Lab, especially Dr. Aniket Ravindra Shirsat, Dr. Zahi Kakish, Dr. Amir Salimi Lafmejani, Dr. Azadeh Doruduchi, Shenbagaraj Kannapiran, and Rakshith Subramanyam, for their invaluable support and feedback at various points throughout my journey.

This endeavor would not have been possible without the support of my family and friends, who have been a constant source of support through thick and thin.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

CHAPTER 1

INTRODUCTION

Worldwide, the percentage of people living in cities is expected to grow from the current estimate of 54% to 66% by 2050 (Petersen, 2019). Based on historical data, even the recent COVID-19 pandemic is unlikely to negatively influence this trend (Florida, 2021). This has led to an increase in the need for infrastructure in urban areas. However, owing to a shortage of workers, the construction industry has been unable to keep up with the demand, dropping the industry's productivity by 50% compared to the 1960s (Cilia, 2019). Robotics and autonomous systems possess immense potential to revolutionize the construction industry (Delgado, 2019). However, the adoption of robotics in the construction industry has been relatively low owing to several factors, such as high initial capital costs, unskilled workforce, and low R&D budgets, with high initial capital costs being the prime factor (Delgado, 2019). Thus, there is a clear need to reduce the high initial capital costs associated with adopting robotic technologies in the construction industry.

Robots deployed in construction applications must operate reliably in the presence of the inherent uncertainties associated with a construction environment and work safely alongside humans, thus, giving rise to the need for semantically driven decision-making (Garg, 2020). According to Garg et al. (2020), semantics in a robotics context is defined as the ability of the robot to understand the meaning of places, objects, or other entities in its surroundings, which is essential for developing intelligent behaviors and safe human-robot collaboration. Such decision-making abilities can be realized by coupling visual sensing with learning-based methods, which is a rapidly growing field of research and has seen increased research output in recent times (Garg, 2020).

Monocular cameras are one of the most widely-used visual sensing solutions for mobile robots owing to their small form factor, low cost, and suitability for agents with space and power constraints. Depth estimation using a monocular camera is a research problem that has gained significant traction in recent years owing to advances in computer vision (Mertan, 2022). With improvements in techniques for depth estimation using monocular cameras, the need for more expensive depth sensors such as Lidar and other time-of-flight sensors could potentially be eliminated.

Thus, to accelerate the adoption of robotic technologies in the construction industry, there is a clear need to develop robot control strategies that use measurements from low-cost sensors, rely on little to no prior knowledge of the environment, and can make decisions based on context. Such controllers could potentially reduce the cost of the robots used and enable them to work in dynamic construction environments.

This thesis aims to leverage advances in the field of deep learning to develop a vision-based controller where the only sensory input to the controller is obtained from a single monocular camera. The controller uses no other inputs, including any prior information about the environment. The controller is tested on a small mobile robotic manipulator in simulation for a material retrieval task in which the robot needs to repeatedly search for a block in a repository zone, pick up the block, and transport it to a deposition zone. The goal of this thesis is to demonstrate the effectiveness of the controller at this material retrieval task. Such an approach, in which the controller works with sensor information from low-cost sensors such as monocular cameras, could potentially reduce the cost of robots used while facilitating semantically-driven decisions.

1.1 Literature Survey

Although the earliest works in robotic construction, in the form of robotic bricklaying, appeared about 25 years ago, the field has seen limited commercial usage and adaptations (Lussi, 2018). As mentioned in the previous section, high capital costs, aversion to change, and an unskilled workforce are some of the biggest challenges facing the adoption of robotics technologies in the construction industry (Delgado, 2019). However, an increased volume of research in the area lately points to a growing interest in the field among researchers. Competitions such as the Mohammed Bin Zayed International Robotics Challenge (MBZIRC) (*MBZIRC*, 2022) have fueled this growth in the interest of researchers. Mobile manipulators are suitable for construction applications owing to their large configuration spaces and ability to maneuver in highly dynamic and unstructured environments. Some of the works in the field of robotics for construction are summarized in the following paragraphs.

This paragraph discusses some of the works in the field of on-site autonomous construction using robots. Jung et al. (2014) demonstrated the use of a humanoid robot for autonomous on-site floor tiling of mosaics. The tiles were identified using an object recognition algorithm that performed edge detection and color detection by calculating the position of the tile in the coordinate frame of the robot's camera. The positions of the robot's joints, relative to the camera frame of reference, were calculated using the robot's kinematics. The robot was able to pick up the tiles using a suction gripper. Feng et al. (2014) utilized a 7 Degree-of-Freedom (DOF) robot manipulator for autonomous robotic assembly in unstructured construction sites. The manipulator was localized with respect to the material repository and deposition zones by utilizing AprilTags (Wang, 2016) visualized through a monocular camera. The resulting pose estimates were used for planning the motion of the manipulator. Feng et al. (2015) expanded their previous work to include a 3D camera on the manipulator to generate an as-built Building Information Model (BIM) using the point clouds generated by the camera. Lussi et al. (2018) utilized a mobile manipulator to construct a full-scale, load-bearing, steel-reinforced concrete wall. The robot was manually repositioned using a joystick for every meter of wall constructed. Localization of the robot and the end-effector were achieved autonomously using AprilTags, which were detected by a wide-baseline stereo camera mounted on the arm. The system also included an on-the-fly building planner, which used images from the stereo camera mounted on the arm to compensate for material deflections during the welding process.

Basri et al. (2021) and Štibinger et al. (2021) developed a mobile manipulator to compete in the MBZIRC 2020, which involved building a predefined structure in an outdoor environment. The robots were required to autonomously navigate, identify the materials, and pick up and place them at different locations to build the predetermined structure. Three Unmanned Aerial Vehicles (UAVs) were used to identify and communicate the position of the building materials to the ground robots. For robot localization and waypoint navigation, Basiri et al. (2021) combined the IMU data, wheel odometry, and GPS in combination with an Extended Kalman Filter (EKF) to obtain the estimated robot pose. In contrast, Štibinger et al. (2021) used the Adaptive Monte Carlo Localization (AMCL) method coupled with LiDAR data for robot localization and waypoint navigation. In both works, the manipulator uses an eye-in-hand approach to detect and pick up the blocks.

In recent years, various vision-based control approaches have been developed for mobile robots used in construction tasks. Asadi et al. (2018) developed a vision-based controller for the real-time application of Unmanned Vehicles (UVs) in construction environments. The controller utilized a monocular camera-based localization and context awareness system. The former was achieved using ORB-SLAM (Mur-Artal, 2015), and the latter was achieved using a lightweight neural network, ENet (Paszke, 2016). The two units worked in parallel to determine the feasible paths for the UVs. Asadi et al. (2021) built on their previous work by integrating a manipulator on the UV. The monocular camera was replaced with a set of two stereo cameras, one for context awareness and localization and the other for visualizing and estimating the block's position. ORB-SLAM was replaced with RTAB-MAP (Labbé, 2019), and the neural network architecture was changed to LNSNet (Asadi, 2019). Tsai et al. (2020) developed a deep convolutional neural network-based control architecture using imitation learning for controlling an omnidirectional mobile manipulator. The controller uses visual data from a stereo camera to drive toward and pick up objects of interest and achieves an average success rate of 78.2% for picking up the objects.

Some of the other approaches used for controlling mobile manipulators outside of applications in construction, as outlined in the literature, are as follows. Sandakalum et al. (2022), in their review of motion planning techniques used for mobile manipulators, identify two primary approaches used for controlling the system: *Decoupled* and *Unified*. In the *Decoupled* approach, the mobile base and the manipulator control are considered independently, and the task execution is achieved sequentially. First, the base is driven to the goal position, and then the arm is used for object manipulation. Some common algorithms used in this approach for both the base and the arm are A*, RRT, RRT-Connect, genetic algorithms, or their variants. As the name suggests, the *Unified* approach considers a coupled model of both the base and arm of a mobile manipulator and is computationally more expensive. Some of the algorithms used for controlling robots that are modeled using the *Unified* approach include CHOMP (Ratliff, 2009) and STOMP (Kalakrishnan, 2011). Iriondo et al. (2019), in their review of Deep Reinforcement Learning (DRL) for pick-and-place operations in logistics, successfully implemented algorithms such as Proximal Policy Optimization (PPO) and Deep Deterministic Policy Gradient (DDPG) on mobile manipulators for learning pick-and-place tasks. Similarly, Wang et al. (2020) demonstrated the effectiveness of PPO for mobile manipulation by implementing it on an experimental platform.

Collective robot construction using swarms of low-cost robots, which can be deployed autonomously with minimal or no human oversight to build structures far larger than themselves, is a promising technique for catering to the increased demand of the construction industry (Werfel, 2012) (Petersen, 2017) (Petersen, 2019). In addition, swarm foraging is an application that has been researched for a long time (Lu, 2020) and could potentially be employed in construction applications for material retrieval and transportation tasks.

1.2 Contribution of the Thesis

In this thesis, we develop a novel vision-based controller for a nonholonomic differential-drive mobile robot equipped with a manipulator arm, which can be used for material retrieval tasks in construction applications. This controller differs from existing in

that it uses a monocular camera as the primary sensor and does not require prior information about the environment. The controller integrates existing deep-learning-based object detection and depth estimation algorithms, using images obtained from a single monocular camera as the only sensor input. The controller can be deployed on different types of mobile manipulator platforms with minimal modifications. We test the controller in the Gazebo 3D robot simulator (Koenig, 2004) for scenarios in which a robot must autonomously search for, identify, pick up, transport, and deposit individual blocks that represent construction material. The controller was simulated for 50 trials, and the success rate of the controller was 46%. This thesis presents a thorough failure analysis for the controller and proposes solutions to address the different errors.

CHAPTER 2

MATHEMATICAL MODELING OF A MOBILE MANIPULATOR

2.1 Introduction to Mobile Manipulators (MMs)

Mobile manipulators (MMs) are robotic systems consisting of articulated arms (manipulators) mounted on holonomic or non-holonomic mobile platforms (Tzafestas, 2014). A mobile manipulator combines a manipulator's dexterity with a mobile robot's mobility, thus creating a platform with far superior capabilities than either one considered individually (Tzafestas, 2014). Depending on the number of Degrees of Freedom (DOF) of the mobile base and the manipulator, a mobile manipulator could be kinematically redundant. Typically, mobile manipulators are kinematically redundant, i.e., they have more than 6 DOF (Sandakalum, 2022). A kinematically redundant mobile manipulator can realize the same end-effector pose in more than one configuration, thus providing more flexibility in terms of the configurations that can be achieved. Mobile manipulators are used in a wide array of applications, including but not limited to medicine, military, construction, and space exploration (Sandakalum, 2022).

In this thesis, a decoupled mathematical model of a mobile manipulator, in which the mathematical models of the manipulator and the base are defined independently, is used to control the mobile manipulator. This decoupled model suffices for our control objectives since we do not require complex robot configurations, and the manipulator is controlled in an open-loop manner; moreover, a coupled (i.e., unified) model would be more computationally expensive to simulate. The following section discusses the kinematics and dynamics of the mobile manipulator.

2.2 Homogeneous Transformations

The following derivation of the homogeneous transformation of an arbitrary point $\boldsymbol{P}$ in space (Figure 1) is adopted from (Siciliano, 2012).

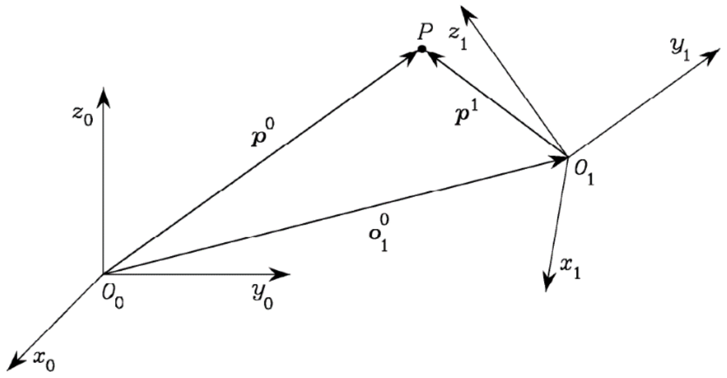


Figure 2.1: Point $\boldsymbol{P}$ in Space Defined With Respect to a Local and Fixed Frame (Siciliano, 2012).

Consider the following notations,

$\boldsymbol{p^0}$ is the position $[\boldsymbol{x_0}, \boldsymbol{y_0}, \boldsymbol{z_0}]^T$ vector of the point $\boldsymbol{P}$ with respect to a fixed reference frame, $\boldsymbol{O_0}$.

$\boldsymbol{o_1^0}$ is the position vector describing the origin of Frame 1 with respect to Frame 0.

$\boldsymbol{p^1}$ is the position vector $[\boldsymbol{x_1}, \boldsymbol{y_1}, \boldsymbol{z_1}]^T$ of the point $\boldsymbol{P}$ with respect to a fixed reference frame, $\boldsymbol{O_1}$.

$\boldsymbol{R_1^0}$ is the rotation matrix describing the orientation of Frame 1 with respect to Frame 0

Thus, the position of the point with respect to the base frame can be defined as follows:

$$\boldsymbol{p^0} = \boldsymbol{o_1^0} + \boldsymbol{R_1^0}\boldsymbol{p^1} \tag{2.1}$$

We know that any point in a three-dimensional space is represented as a (3×1) vector, with the components representing the Cartesian coordinates of that point in space. To achieve a compact representation of the same point defined in two different frames as in equation 2.1, we adopt the homogeneous representation of a vector. By adding a fourth unit component to a generic vector $\boldsymbol{p}$, we get $\boldsymbol{p'}$ which is defined as follows:

$$\boldsymbol{p'} = \begin{bmatrix} \boldsymbol{p} \\ 1 \end{bmatrix} \tag{2.2}$$

Thus, by adopting the above notation for $\boldsymbol{p^0}$ and $\boldsymbol{p^1}$ the coordinate transformation can be written as,

$$\boldsymbol{A_1^0} = \begin{bmatrix} \boldsymbol{R_1^0} & \boldsymbol{o_1^0} \\ \boldsymbol{0^T} & 1 \end{bmatrix} \tag{2.3}$$

where, $\mathbf{0^T}$ is a (1×3) vector of zeros, $\mathbf{o_1^0}$ is the (3×1) translation vector of Frame 1 with respect to Frame 0 and $\mathbf{R_1^0}$ is as defined earlier.

2.3 Denavit-Hartenberg (DH) Parameters

The Denavit-Hartenberg parameters are used to define the relative position of two consecutive links in a serial manipulator (Siciliano, 2012). To achieve this, the convention lays out a set of basic rules. The following rules and nomenclature have been adopted from (Siciliano, 2012). Consider the following assumption:

$\boldsymbol{i}$ is the axis of the joint connecting link $\boldsymbol{i-1}$ to link $\boldsymbol{i}$, as shown in Figure 2.2.

Then, the Denavit-Hartenberg convention is defined as follows:

- Choose axis $\boldsymbol{z_i}$ along the axis of Joint $\boldsymbol{i-1}$.
- Locate the origin $\boldsymbol{O_i}$ at the intersection of the axis $\boldsymbol{z_i}$ with the common normal to axes $\boldsymbol{z_{i-1}}$ and $\boldsymbol{z_i}$. Locate $\boldsymbol{O_{i'}}$ at the intersection of the common normal with axis $\boldsymbol{z_{i-1}}$.

- Choose axis $\boldsymbol{x_i}$ along the common normal to axes $\boldsymbol{z_{i-1}}$ and $\boldsymbol{z_i}$ with direction from Joint $\boldsymbol{i}$ to $\boldsymbol{i+1}$.
- Choose axis $\boldsymbol{y_i}$ to complete a right-handed frame of reference.

Using the DH Parameters, it is possible to calculate the homogeneous transformation matrix between links $\boldsymbol{i-1}$ and $\boldsymbol{i}$ as in equation (2.3), which can be repeated for all links to calculate the forward kinematics of the robot arm.

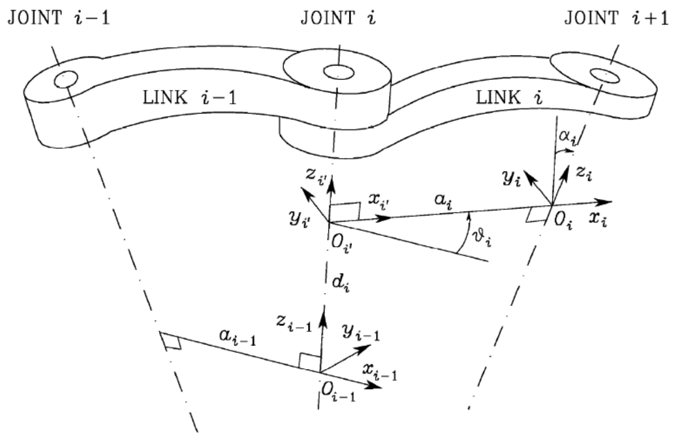


Figure 2.2: Denavit-Hartenberg Convention (Siciliano, 2012).

However, the convention provides a non-unique definition of the link frames in certain cases and configurations of a manipulator (Siciliano, 2012). These have not been discussed in this thesis. The reader is encouraged to refer to the cited resource for more information.

2.4 Kinematics and Dynamics of a Differential Drive Wheeled Mobile Robot (WMR)

The following derivation has been adopted from (Tzafestas, 2014). Consider a differential drive WMR as shown in Figure 2.3 with the following nomenclature:

$\boldsymbol{Q}$ is the midpoint of the line connecting the two driven wheels and $\boldsymbol{v_Q}$ is its velocity.

$\boldsymbol{b}$ is the distance of $\boldsymbol{Q}$ from the center of gravity $\boldsymbol{G}$ of the robot.

$\boldsymbol{2a}$ is the width of the robot.

$(\boldsymbol{x_Q}, \boldsymbol{y_Q})$ and $(\dot{\boldsymbol{x}}_{\boldsymbol{Q}}, \dot{\boldsymbol{y}}_{\boldsymbol{Q}})$ are the position and velocity coordinates of point Q, respectively.

$\boldsymbol{\phi}$ is the heading angle of the robot.

$\boldsymbol{v_r}$ and $\boldsymbol{v_l}$ are the linear velocities of the right and left wheels, respectively.

$(\boldsymbol{\theta_r}, \boldsymbol{\theta_l})$ and $(\dot{\boldsymbol{\theta}}_{\boldsymbol{r}}, \dot{\boldsymbol{\theta}}_{\boldsymbol{l}})$ are the angular positions and speeds of the right and left wheels, respectively.

$\boldsymbol{r}$ is the radius of the wheels.

The following assumptions are made:

- The wheels roll without any slippage
- The steering axis is perpendicular to the $\boldsymbol{x-y}$ plane
- The point $\boldsymbol{Q}$ coincides with the center of gravity $\boldsymbol{G}$

Then, the kinematic model of the WMR can be described by the following equations,

$$\dot{\boldsymbol{x}}_{\boldsymbol{Q}} = \frac{\boldsymbol{r}}{\boldsymbol{2}}\left(\dot{\boldsymbol{\theta}}_r \, \boldsymbol{cos}\, \boldsymbol{\phi} + \dot{\boldsymbol{\theta}}_l \, \boldsymbol{cos}\, \boldsymbol{\phi}\right) \tag{2.4}$$

$$\dot{\boldsymbol{y}}_{\boldsymbol{Q}} = \frac{\boldsymbol{r}}{\boldsymbol{2}}\left(\dot{\boldsymbol{\theta}}_r \, \boldsymbol{sin}\, \boldsymbol{\phi} + \dot{\boldsymbol{\theta}}_l \, \boldsymbol{sin}\, \boldsymbol{\phi}\right) \tag{2.5}$$

$$\dot{\boldsymbol{\phi}} = \frac{\boldsymbol{r}}{\boldsymbol{2a}}\left(\dot{\boldsymbol{\theta}}_r - \dot{\boldsymbol{\theta}}_l\right) \tag{2.6}$$

with the non-holonomic constraints given as follows:

$$\boldsymbol{M}\dot{\boldsymbol{p}} = \boldsymbol{0} \tag{2.7}$$

where,

$$\boldsymbol{M} = [-\boldsymbol{sin\,\phi} \quad \boldsymbol{cos\,\phi} \quad \mathbf{0}] \tag{2.8}$$

and,

$$\dot{\boldsymbol{p}} = \begin{bmatrix} (\boldsymbol{r}/2)\,\boldsymbol{cos\,\phi} \\ (\boldsymbol{r}/2)\,\boldsymbol{sin\,\phi} \\ \boldsymbol{r}/2\boldsymbol{a} \end{bmatrix} \dot{\boldsymbol{\theta}}_{\boldsymbol{r}} + \begin{bmatrix} (\boldsymbol{r}/2)\,\boldsymbol{cos\,\phi} \\ (\boldsymbol{r}/2)\,\boldsymbol{sin\,\phi} \\ -\boldsymbol{r}/2\boldsymbol{a} \end{bmatrix} \dot{\boldsymbol{\theta}}_{\boldsymbol{l}} \tag{2.9}$$

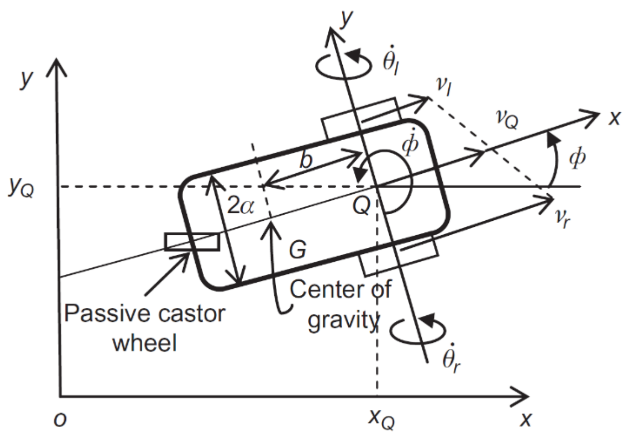


Figure 2.3: Differential Drive WMR (Tzafestas, 2014).

The dynamics of differential drive WMR is given by (Tzafestas, 2014):

$$\dot{\boldsymbol{v}} = \frac{\mathbf{1}}{\boldsymbol{mr}}(\boldsymbol{\tau}_{\boldsymbol{r}} + \boldsymbol{\tau}_{\boldsymbol{l}}) \tag{2.10}$$

$$\dot{\boldsymbol{\omega}} = \frac{\mathbf{2}\boldsymbol{a}}{\boldsymbol{Ir}}(\boldsymbol{\tau}_{\boldsymbol{r}} - \boldsymbol{\tau}_{\boldsymbol{l}}) \tag{2.11}$$

Where, $\boldsymbol{\tau}_{\mathbf{r}}$ and $\boldsymbol{\tau}_{\mathbf{l}}$ are the torques on the right and left wheels due to the corresponding forces $\mathbf{F}_{\mathbf{r}}$ and $\mathbf{F}_{\mathbf{l}}$, respectively; **I** is the inertia matrix; **m** is the mass of the robot; **r** and **a** are as defined earlier**.** The reader is encouraged to refer to the cited reference for the derivation of equations (2.10) and (2.11).

2.5 Kinematics and Dynamics of a Serial Manipulator

The kinematic analysis of a Serial Manipulator consists of three parts: Forward Kinematics, Inverse Kinematics, and Differential Kinematics. Forward Kinematics involves the task of finding the end-effector position, given a certain joint configuration of the robot arm. Consider the following nomenclature:

Let $\boldsymbol{b}$ and $\boldsymbol{e}$ denote the base frame and the end-effector frame of the robot respectively. Let $0 \ldots \boldsymbol{n}$ denote the intermediate frames corresponding to the joints between $\boldsymbol{b}$ and $\boldsymbol{e}$. Thus, using equation (2.3) and the DH convention, the direct kinematics function of the serial manipulator which computes the end-effector pose with respect to the base frame can be defined as follows (Siciliano, 2012):

$$\boldsymbol{A}_{\boldsymbol{e}}^{\boldsymbol{b}} = \boldsymbol{A}_{\boldsymbol{0}}^{\boldsymbol{b}} \boldsymbol{A}_{\boldsymbol{n}}^{\boldsymbol{0}} \boldsymbol{A}_{\boldsymbol{e}}^{\boldsymbol{n}} \tag{2.12}$$

In Differential Kinematics the goal is to find the relationship between the joint velocities and the end-effector linear and angular velocities (Siciliano, 2012).

Consider the following nomenclature (Siciliano, 2012):

$\boldsymbol{q} = [q_1 \quad q_2 \cdots q_{n-1} \quad q_n]^T$ be the vector of joint variables.

$\dot{\mathbf{p}}_{\boldsymbol{e}}$ and $\boldsymbol{\omega}_{\boldsymbol{e}}$ be the linear and angular velocity of the end-effector respectively.

$\boldsymbol{J}_{\boldsymbol{P}}(\boldsymbol{q})$ and $\boldsymbol{J}_{\boldsymbol{O}}(\boldsymbol{q})$ be the $(3 \times n)$ Jacobian matrices relating the contribution of the joint velocities to the end-effector linear and angular velocities respectively.

Thus, the following definitions can be used to determine the relationship between $\dot{\mathbf{p}}_{\boldsymbol{e}}$ and $\boldsymbol{\omega}_{\boldsymbol{e}}$ (Siciliano, 2012):

$$\dot{\boldsymbol{p}}_{\boldsymbol{e}} = \boldsymbol{J}_{\boldsymbol{P}}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{2.13}$$

$$\boldsymbol{\omega_e} = \boldsymbol{J_O}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{2.14}$$

In Inverse Kinematics the goal is to find the configuration $\boldsymbol{q}$ of the robot arm given an end-effector pose (Tzafestas, 2014). The solution to the inverse kinematics problem is not unique; and depending on the number of degrees of freedom of the system, numerous solutions are possible (Siciliano, 2012). Both analytical and numerical methods have been developed for calculating the inverse kinematics of a serial manipulator. The reader is encouraged to refer to the cited works for a comprehensive treatment of the topic.

The dynamics of a serial manipulator are typically derived using the Lagrange method and the reader is encouraged to refer to the cited works for a thorough treatment of the topic.

2.6 Kinematics and Dynamics of a Mobile Manipulator

The following equations and nomenclature for the kinematics and dynamics of the mobile manipulator have been adopted from (Tzafestas, 2014).

Consider the following four coordinate frames:

$\boldsymbol{O_w x_w y_w z_w}$ is the world coordinate frame.

$\boldsymbol{O_p x_p y_p z_p}$ is the platform coordinate frame.

$\boldsymbol{O_b x_b y_b z_b}$ is the coordinate frame of the manipulator's base**.**

$\boldsymbol{O_e x_e y_e z_e}$ is the coordinate frame of the manipulator's end-effector.

$\boldsymbol{x_e^w}$ is the end-effector pose in the world coordinate frame.

$\boldsymbol{u}(\boldsymbol{t}) = \left[u_p^T(t), u_m^T(t)\right]^T$ is the vector of control commands to the platform, and the end-effector respectively.

$\boldsymbol{p} = [x \quad y \quad \phi]^T$ represents the platform configuration.

$\boldsymbol{\theta} = [\theta_1 \quad \theta_2 \quad \theta_3 \cdots \theta_n]^T$ represents the manipulator configuration.

$\boldsymbol{q} = [\boldsymbol{p}^{\boldsymbol{T}}, \boldsymbol{\theta}^{\boldsymbol{T}}]^{\boldsymbol{T}}$ denotes the combined configuration of the WMR and manipulator.

$\boldsymbol{J}(\boldsymbol{q})$ be the combined Jacobian of the WMR and manipulator.

So, the transformation from the world coordinate frame to the end-effector coordinate frame is given as follows using equation (2.3):

$$\boldsymbol{A}_{\boldsymbol{e}}^{\boldsymbol{w}} = \boldsymbol{A}_{\boldsymbol{p}}^{\boldsymbol{w}} \boldsymbol{A}_{\boldsymbol{b}}^{\boldsymbol{p}} \boldsymbol{A}_{\boldsymbol{e}}^{\boldsymbol{b}} \tag{2.15}$$

And the overall kinematic model of the mobile manipulator is given as follows:

$$\dot{\boldsymbol{x}}_{\boldsymbol{e}}^{\boldsymbol{w}} = \boldsymbol{J}(\boldsymbol{q})\boldsymbol{u}(\boldsymbol{t}) \tag{2.16}$$

subject to the nonholonomic constraint,

$$\boldsymbol{M}(\boldsymbol{p})\dot{\boldsymbol{p}} = 0, \;\; \boldsymbol{M}(\boldsymbol{p}) = [\dot{sin}\,\phi \quad cos\,\phi \quad 0 \cdots 0] \tag{2.17}$$

The dynamics of the mobile manipulator are derived using the Lagrange method and the reduced (unconstrained) model that describes the dynamic evolution of $\boldsymbol{q}(\boldsymbol{t})$ in terms of the dynamic evolution of $\boldsymbol{v}(\boldsymbol{t})$, is as follows:

$$\overline{\boldsymbol{D}}(\boldsymbol{q})\dot{\boldsymbol{v}} + \overline{\boldsymbol{C}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\boldsymbol{v} + \overline{\boldsymbol{g}}(\boldsymbol{q}) = \overline{\boldsymbol{E}}\boldsymbol{\tau} \tag{2.18}$$

For the definitions of the matrices $\overline{\boldsymbol{D}}, \overline{\boldsymbol{C}}, \overline{\boldsymbol{g}}$ and $\overline{\boldsymbol{E}},$ the reader is encouraged to refer to references (Tzafestas, 2014, Siciliano, 2012 and Lynch, 2019).

CHAPTER 3

BASIC CONCEPTS IN COMPUTER VISION

3.1 What is computer vision?

Computer vision is the enterprise of automating and integrating a wide range of processes and representations used for visual perception and is the inverse problem of computer graphics (Szeliski, 2011) (Ballard, 1981). The task is to describe the world we see in one or more images and reconstruct its properties, such as shape, illumination, and color distributions (Szeliski, 2011). From the engineering point of view, the goal is to build autonomous systems which can perform or outperform some of the tasks that the human visual system can do (Huang, 1996). Computer vision is widely used today in a wide variety of real-world applications such as photogrammetry, automotive applications, medicine, motion capture, and 3D modeling (Szeliski, 2011). It is an umbrella term that encompasses multiple tasks such as image processing, object detection, and pattern classification (Ballard, 1981). Some of the topics relevant to this thesis have been discussed briefly in the subsequent sections.

3.2 Object Detection

The object detection problem involves two major tasks: (1) to determine the location of objects in each image, also called object localization, and (2) to determine which category each object belongs to, also called object classification (Zhao, 2019). An example of an object detection task has been illustrated in Figure 3.1. An object detection pipeline has three major stages: Informative region selection, Feature extraction, and Classification (Zhao, 2019). A brief description of the tasks involved in the three stages are as follows:

(1) ***Informative region selection****:* An image can contain several objects located at different positions, which could be of different sizes or aspect ratios (Zhao, 2019). The goal of informative region selection is to use a multi-scale sliding window to extract information about the positions of different objects in the image (Zhao, 2019). Although this method converges, i.e., it can extract the positional information of all objects in the image, it is computationally expensive and produces too many redundant windows (Zhao, 2019). ***(2) Feature extraction:*** The next step is to extract information about an image's visual features, providing semantics and robust representations, which are then used to identify the different objects within an image (Zhao, 2019). Scale Invariant Feature Transform (SIFT) is one of the methods used in feature extraction (Lowe, 2004). ***(3) Classification:*** The last task in object detection is the task of classification, where the goal, as the name suggests, is to classify the image or the objects in the image into one or more predefined categories (Zhao, 2019). This step makes the representations of the objects in the image more hierarchical, semantic, and informative for visual recognition (Zhao, 2019).

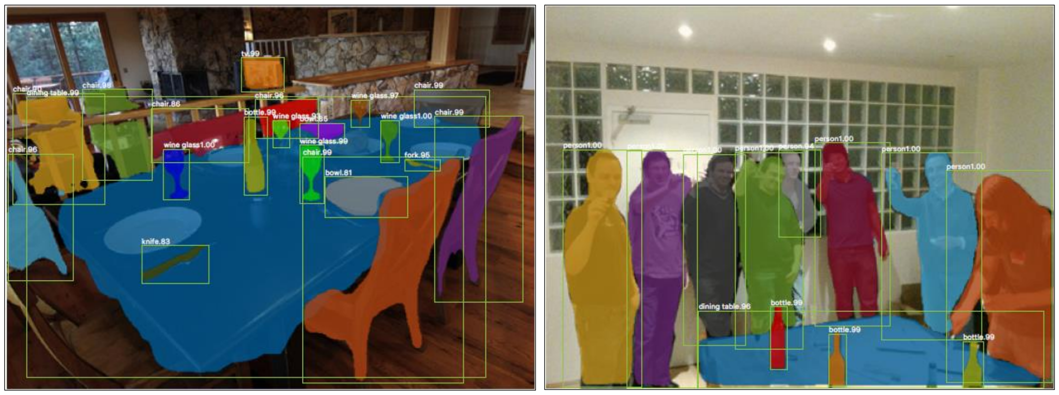


Figure 3.1: Object Detection Examples (He, 2017)

3.3 Depth Estimation

Estimating the depth information from 2D images is one of the most critical tasks in computer vision (Zhao, 2020). It involves evaluating a dense depth map for a given RGB image, i.e., one must calculate a metric depth value for each pixel in the given RGB image (Mertan, 2022). It is an essential component of understanding the geometric relationships within a scene and is used in many applications such as SLAM, navigation, and object detection (Eigen, 2014) (Zhao, 2020). An example of depth estimation from a pair of stereo images is shown in Figure 3.2, where the shading in the image becomes darker as the distance from the camera increases, and the unknown regions are represented in black (Scharstein, 2003).

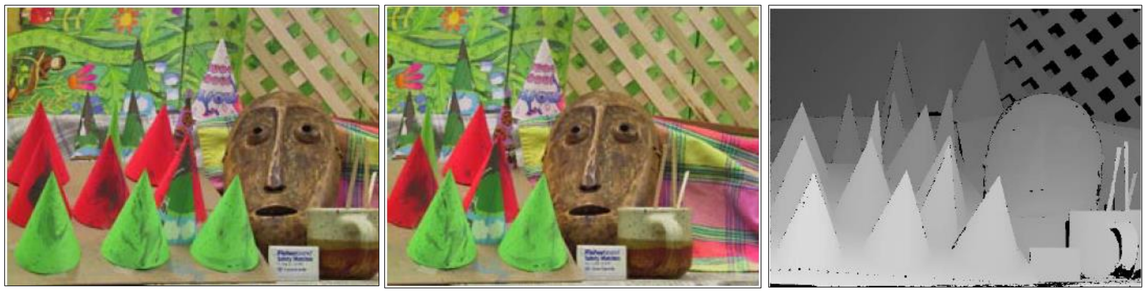


Figure 3.2: Example of Depth Estimation Using Stereo Images, Left Camera Image (Left), Right Camera Image (Center), and Final Disparity Map (Right) (Scharstein, 2003)

There are three primary methods for estimating depth from RGB images: Geometry-based methods, Sensor-based methods, and Deep learning-based methods (Zhao, 2020). ***(1) Geometry-based methods*** are widely popular within the research community and have been studied for the past forty years (Zhao, 2020). One of the most widely used techniques is to estimate depth using a pair of stereo images (Zhao, 2020). A stereo image is a pair of images captured using two cameras (left and right) separated by a fixed distance. Each camera receives a slightly different view of the world, and this disparity between the images is exploited to extract information about the image (Saxena, 2007). These disparities vary inversely with the object's distance from the camera, and small changes in disparities result in significant errors in the depth estimates (Saxena, 2007). Another example of a geometry-based method is Structure from Motion (SfM) (Ullman, 1979), where feature correspondences and geometric constraints between image sequences are used to obtain the depth of sparse features (Zhao, 2020). ***(2) Sensor-based methods***, as the name suggests, utilize sensors such as RGB-D cameras and LiDAR to directly get the corresponding images' depth information without using geometric estimation techniques (Zhao, 2020). Although RGB-D cameras can provide pixel-level depth maps, their range of operation is limited and they are not robust to intense lighting conditions (Zhao, 2020), while LiDAR can only generate sparse 3D maps. These sensors have a few drawbacks in terms of cost, weight, and size, thus making them unsuitable for power and size-constrained platforms such as small drones (Zhao, 2020). Such issues have led to increased research in depth estimation from monocular cameras using deep learning methods (Zhao, 2020). ***(3) Deep learning-based methods*** have shown incredible potential and breakthrough performance on various computer vision tasks, ranging from image classification to depth estimation (Zhao, 2020). These methods employ neural networks with varying architectures, depending on the task and the accuracy required. The techniques hold great promise for use on systems with limited sensing capabilities, since increasing research in the field could potentially eliminate the need for large, expensive sensors.

3.4 Neural Networks and Deep Learning

According to Mitchell (1997), the definition of machine learning is as follows: "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E." Artificial Neural Networks are popular techniques that simulate the mechanisms of biological organisms and are widely used in machine learning applications (Aggarwal, 2018). The biological neurons are connected using axons and dendrites, and the region between the connections are called synapses (Aggarwal, 2018). An artificial neural network consists of mathematical neurons connected in either a fully connected or sparsely connected manner, and they are designed to mimic the functions of a biological neuron. The perceptron is the simplest neural network and consists of a single input layer and an output node, and the illustration of its architecture is shown in Figure 3.3 (Aggarwal, 2018). The simple mathematical description of a perceptron is as follows (Aggarwal, 2018):

Let $\boldsymbol{d}$ be the number of nodes that transmit the $\boldsymbol{d}$ features.

$\overline{\boldsymbol{X}} = [x_1 \cdots x_d]$ be the feature vector and $\overline{\boldsymbol{W}} = [w_1 \cdots w_d]$ be the corresponding edge weights.

Therefore, the prediction $\hat{y}$ at the output layer is given as follows:

$$\hat{\boldsymbol{y}} = activation\_function\{\sum_{j=1}^{d} w_j x_j\} \tag{3.1}$$

Where the $activation\_function$ is chosen based on the application, e.g., *sign* and *ReLU*, a bias value can be selected to influence the output value as well.

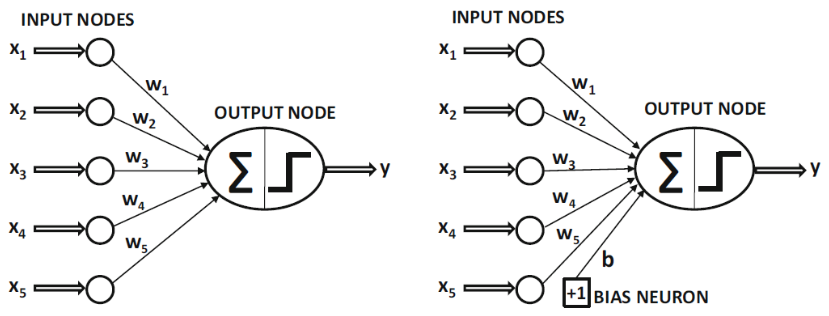


Figure 3.3: Basic Architecture of a Perceptron, Without Bias (Left) and With Bias (Right) (Aggarwal, 2018)

Unlike the perceptron, which contains a single input and output layer as seen earlier, multi-layer neural networks include more than one computational layer (Aggarwal, 2018). A multi-layer neural network has additional layers between the input and output layers; as mentioned earlier, these middle layers are called hidden layers. In multi-layer neural networks, the output from each successive layer feeds into the next layer in the forward direction and are hence also referred to as feed-forward networks (Aggarwal, 2018). Such networks can either be fully or partially connected. However, fully connected architectures perform better in many settings and thus are much more common (Aggarwal, 2018). An illustration of a fully connected feed-forward neural network is shown in Figure 3.4.

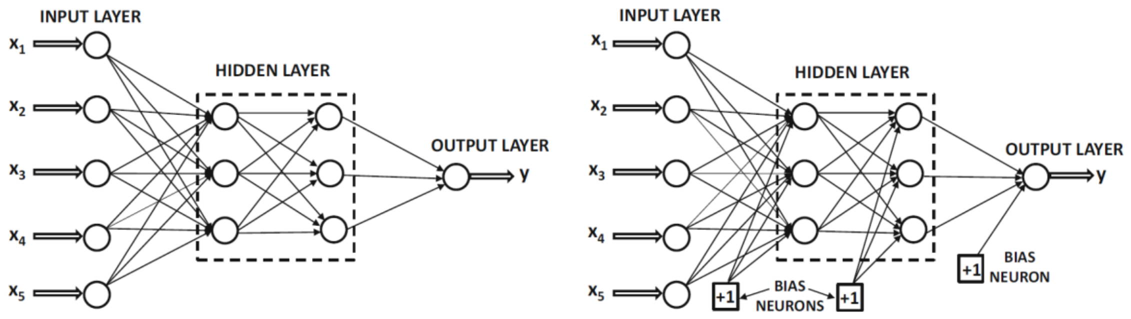


Figure 3.4: Example of a Feedforward Neural Network, Without Bias (Left) and With Bias (Right) (Aggarwal, 2018)

3.5 Depth Estimation and Object detection using Deep Learning

As discussed in earlier sections, the power of deep learning methods to learn features from images using pixel-level relationships and other visual cues has made them suitable for tasks in computer vision, such as depth estimation and deep learning (Zhao, 2020). Several well-known neural network architectures, such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Region-based Convolutional Neural Networks (R-CNNs), and Generative Adversarial Networks (GANs), have been used for the tasks of depth estimation and object detection (Zhao, 2019) (Zhao, 2020). Some newer techniques, such as Mask R-CNN, build on top of Faster R-CNNs by adding a binary mask for each region of Interest (RoI), thus significantly enhancing the performance metrics of the model (He, 2017). An example of depth estimation using a deep neural network-based approach is shown in Figure 3.5. Brighter regions in the output image correspond to parts of the scene that are closer to the camera, and darker regions represent parts that are farther away.

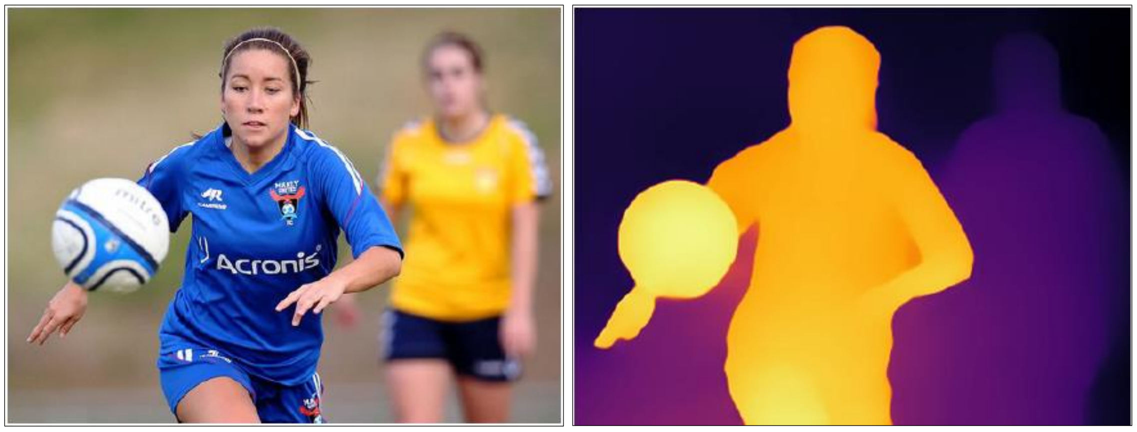


Figure 3.5: Example of Depth Estimation Using Deep Learning (Monocular Cameras), Input Image (Left), Output Image (Right) (Ranftl, 2022)

3.6 What is Transfer Learning

According to Pan (2010), Transfer Learning is defined as, "Given a source domain D_S and learning task T_s, a target domain D_T and learning task T_T, transfer learning aims to help improve the learning of the target predictive function $f_T(.)$ in D_T using the knowledge in D_S and T_S, where $D_S \neq D_T$, or $T_S \neq T_T$". There are three main tasks in transfer learning, *(1) What to transfer* deals with determining which part of the knowledge can be transferred, *(2) How to transfer* deals with the question of how the existing method would be used for the new application, and depends on the task at hand, and *(3) When to transfer* deals with determining the situations and tasks in which transferring should be done (Pan, 2010). An example of the transfer learning task described in Pan (2010) is as follows. Consider a sentiment classifier that classifies product reviews. To build this classifier, we would need to collect a large volume of data, build a model, train it, and tune it to obtain desired output and accuracy. If our data is limited, or if there are errors in data annotations, model's performance would not be as desired. In such cases, transfer learning can improve the model's performance and reduce the overall effort in building the model in general.

3.7 YOLO v5

You Only Look Once (YOLO) is a family of state-of-the-art object detection algorithms, which was initially proposed in 2016 (Redmon, 2016). The model is currently in its fifth iteration (v5), which is a PyTorch implementation of its fourth version (v4) (Jocher, 2020). The model is widely used within the computer vision research community for many real-time computer vision applications. The reason for the model's popularity is its ability to produce high mean average precision (mAP) while maintaining fast runtimes, or frames per second (FPS). Different versions of the model and their properties are as follows:

YOLO v1 utilizes a single neural network to predict bounding boxes and class probabilities directly from full images in one pass and thus, could be optimized directly for detection performance (Redmon, 2016). The model formulates the object detection problem as a regression problem and maps the image pixels directly to bounding box coordinates and probabilities (Redmon, 2016). The model performed extremely well compared to other state-of-the-art models at the time, with 9-40% better Average Precision (AP), depending on the dataset on which it was evaluated (Redmon, 2016). The architecture of YOLO v1 is shown in Figure 3.6.

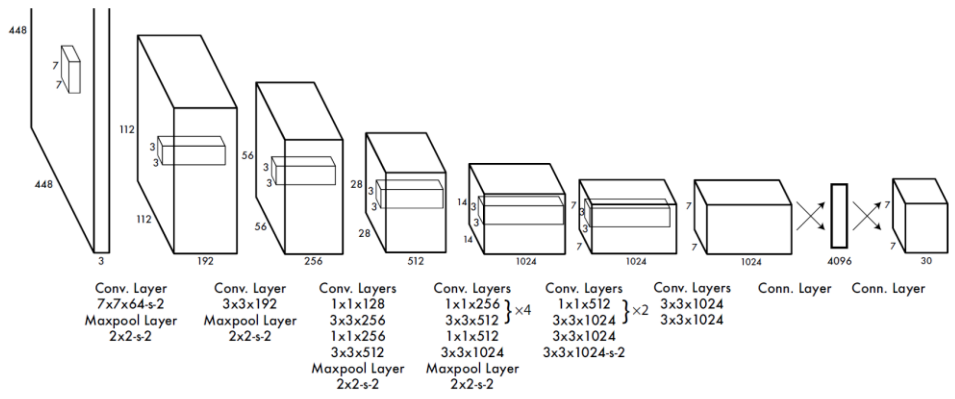
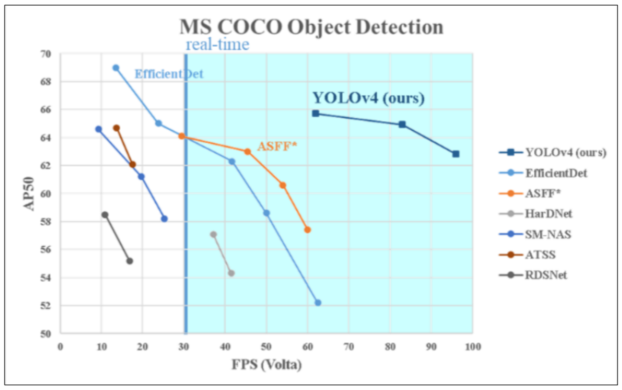


Figure 3.6: Architecture of YOLO v1 (Redmon, 2016)

YOLO v2, also called YOLO9000, is the second iteration in the YOLO family of models. As the name suggests, it could detect over 9000 object categories and used a multi-scale training method, thus enabling it to run at varying sizes, offering a trade-off between speed and accuracy (Redmon, 2017). It also was designed to overcome some of the significant drawbacks of YOLO v1, namely localization errors and low recall compared to region-proposal-based networks (Redmon, 2017). On the VOC 2012 dataset, it achieved a mAP of 73.4% while running at much faster speeds than competing models (Redmon, 2017).

YOLO v3 was an incremental improvement from the previous version, with a newly trained classifier network with added connections to the backbone layer (Redmon, 2018). It had a comparable performance with other state-of-the-art models at the time; as mentioned earlier was simply an incremental improvement over v2.

YOLO v4 implemented a new CSPDarknet53 architecture for the backbone network (Bochkovskiy, 2020). It also adds a PANnet for feature aggregation, SPP for increasing the receptive field, and data augmentation methods ("bag of freebies") which increases the performance of the network without sacrificing inference times (Bochkovskiy, 2020). This resulted in significantly better performance characteristics both in terms of AP and FPS, and the plots of FPS vs. AP50 for different GPU architectures are shown in Figure 3.7.



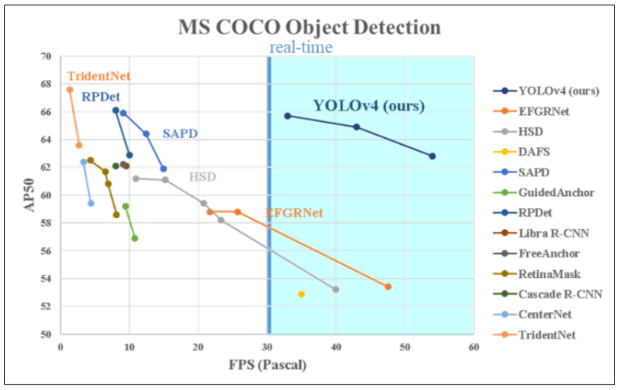


Figure 3.7: YOLO v4 FPS vs. AP50 Plots for Different GPU Architectures on the Microsoft COCO Dataset: Volta (Top), Pascal (Bottom) (Bochkovskiy, 2020)

YOLO v5 is a PyTorch implementation of YOLO v4, is open source, and combines the power of the YOLO model with the flexibility of PyTorch, making it suitable for real-time applications in research (Jocher, 2020). The implementation has a set of pre-trained models with varying sizes with trade-offs in inference speed vs. accuracy. The performance characteristics of the different implementations are shown in Figure 3.8. Refer to Appendix A for the architecture of YOLO v5.

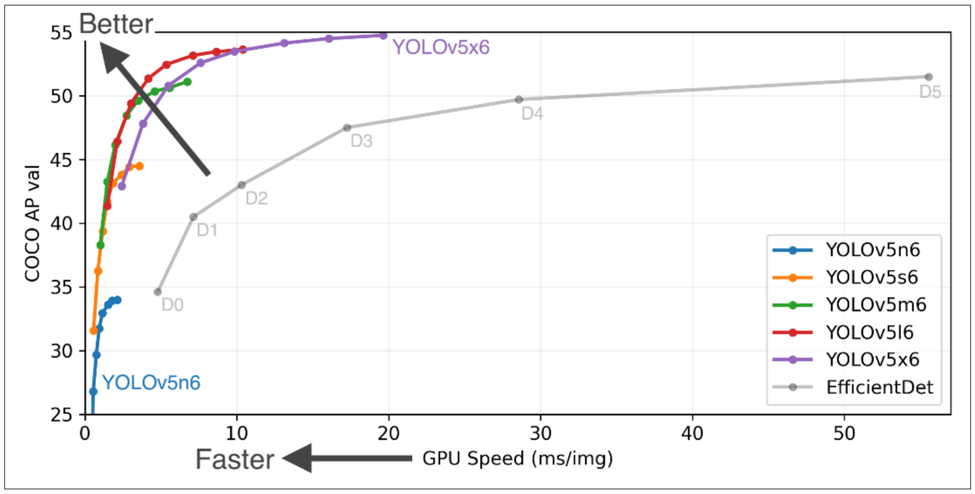


Figure 3.8: Inference Speed vs. AP of YOLO v5 Pretrained Models (Jocher, 2020)

3.8 MiDaS

MiDaS, developed by Ranftl et al. (2022), is a robust monocular depth estimation model that can perform across various environments. The authors trained the model on diverse datasets, using a novel loss function, making the model invariant to major sources of incompatibilities between datasets (Ranftl, 2022). The model trained for about six months on different datasets. The performance was evaluated using the principle of "zero-shot cross dataset transfer," i.e., the model was evaluated on datasets it had never seen before (Ranftl, 2022). The model outperforms other models in the monocular depth estimation using deep learning space, thus making it state-of-the-art. A sample prediction of the model has been illustrated in Figure 3.5. The reader is encouraged to refer to the cited work for details about the model.

CHAPTER 4

CONTROL ARCHITECTURE AND SIMULATION SETUP

This thesis considers a Robotis Turtlebot3 Waffle Pi with an OpenMANIPULATOR-X for the simulations (Figure 4.1). A sequential, decoupled approach has been adopted for controlling the mobile manipulator, i.e., the control of the base and the manipulator are independent of each other. First, the mobile base is controlled using vision-based methods and driven to specific locations in the simulated test environment. Once the mobile base reaches its goal position, the arm controller is activated and is used to pick up a block. The specifics of the individual components of the control architecture, assumptions, and simulation setup are discussed in the following sections.

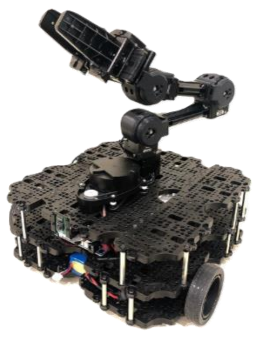


Figure 4.1 Turtlebot3 Waffle Pi With OpenMANIPULATOR X (source: Robotis)

4.1 Base Control

The proposed base control method has three major components, (1) tag-based navigation, (2) object detection, and (3) depth estimation using a Raspberry Pi monocular camera, and they are explained briefly in the following paragraphs.

(1) The ***tag-based navigation system*** consists of AprilTags (Wang, 2016) placed in the Gazebo testbed corresponding to pickup and deposition zones (discussed in later sections). The onboard monocular camera (Raspberry Pi v2) visualizes the tags, and the apriltag_ros package (Wang, 2016) (Malyuta, 2017) computes the pose of the tags with respect to the desired frame of reference on the robot. The robot uses this information for the initial navigation between different zones in the simulated test environment.

(2) The ***object detection algorithm*** was developed by transfer-learning a pre-trained YOLOv5 model to identify blocks belonging to one of five classes: Red Cube, Red Cylinder, Red Sphere, Green Cube, and Green Cylinder. The robot uses its onboard monocular camera to visualize the world around it. The output from the camera is resized to 416×416 pixels, and the neural network uses this input to run inference in real-time. The inference results from the model contain data that includes information about the object's class and bounding boxes. The robot's base controller calculates the object’s centroid coordinates from the inference results and uses them to navigate toward a block of interest.

(3) **D*epth estimation*** using the monocular camera is carried out by employing the MiDaS v3 (Ranftl, 2022) network. Like the object detection algorithm, the depth estimation network runs real-time inference on the resized output images from the camera. The output is a depth map of the image that consists of relative inverse depth values corresponding to each pixel in the image. Thus, as mentioned earlier, combining the outputs from the object detection algorithm and the depth estimation

network, we can identify the depth corresponding to the centroid of the object of interest.

Controlling the robot's base to navigate the different zones in the test environment is achieved using a sequential combination of the three components. The robot uses AprilTags for initial navigation toward the Pickup and deposition zones. Once the robot reaches a predetermined distance threshold relative to the tags, it switches over to the deep-learning-based methods for navigating closer to a block. The object detection algorithm returns the centroid ($centroid\_x$ and $centroid\_y$) of the block, which is used to obtain the corresponding depth value and is further used to calculate the angular velocity command for the mobile base using the following equation:

$$\omega_z = (213 - centroid\_x)/45 \tag{4.1}$$

where ω_z is the angular velocity in rad/s of the mobile base w.r.t the z-axis of the world frame, pixel value 213 corresponds to the midpoint of the image, and 45 is a scaling factor. The resulting value of ω_z is checked to ensure that it is within predetermined limits (± 0.05 rad/s) to ensure safe operation of the system. Section 4.3 describes the equations for calculating the velocity commands in further detail. The velocity commands generated by the control equations are realized on the robot using a standard PID implementation.

4.2 Arm Control

The arm control for the system uses the MoveIt Motion Planning Framework (Coleman, 2014), which is implemented on the Open Motion Planning Library (OMPL) (Șucan,2012) backend. The planner chosen for this application is RRT-Connect. The arm is programmed to reconfigure into predetermined sets of joint angles to pick up and place objects in the test environment. The arm's control system is an open-loop controller, i.e., the arm does not have sensing capabilities to ensure block pickup and deposition. Instead, we rely on the accuracy of the base controller to drive the robot to the required pose (described in Figure 4.3(b)) to ensure that the arm picks up the block. Although this method works well in simulation, a real-world implementation might require the following modifications: (1) an additional camera on the arm and a closed-loop controller with vision-based feedback from this camera to ensure block pickup, or (2) a neural network that is trained to determine whether the block has been picked up using images from the camera mounted on the mobile base.

4.3 Complete Control Architecture

The overall control architecture of the robot is set up as a finite-state machine, shown in Figure 4.2, that incorporates the control approaches discussed in Sections 4.1, 4.2 and 4.4. The implementation of the state machine is realized using the ROS *smach* package. The state machine has the following five states: (1) Navigate towards the Material repository, (2) Navigate to a block, (3) Pick up a block, (4) Navigate to the Construction zone, and (5) Place a block.

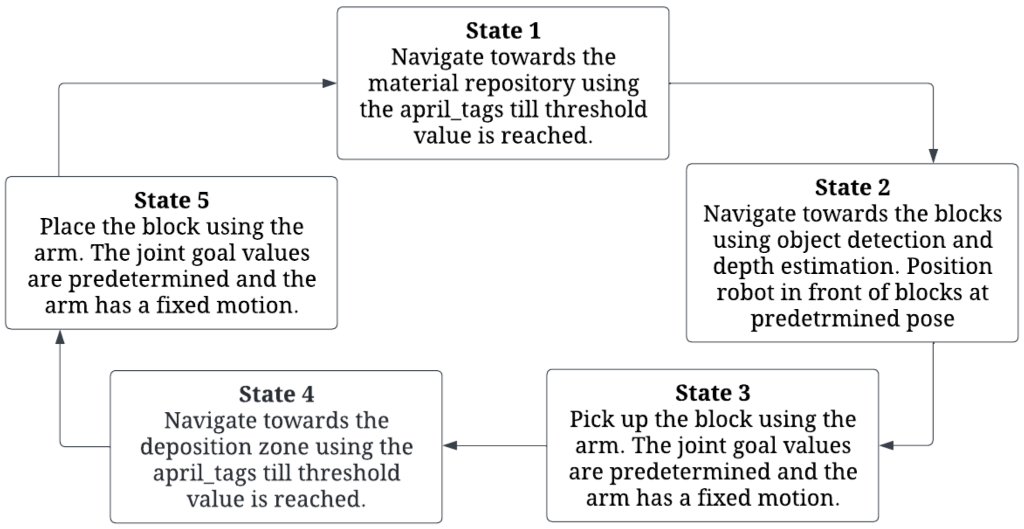
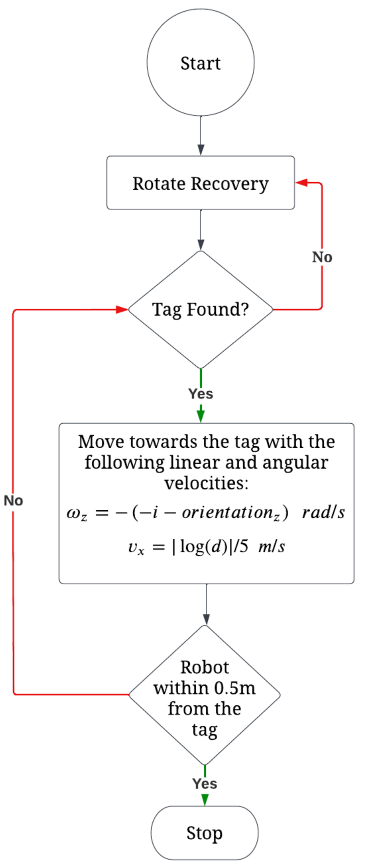
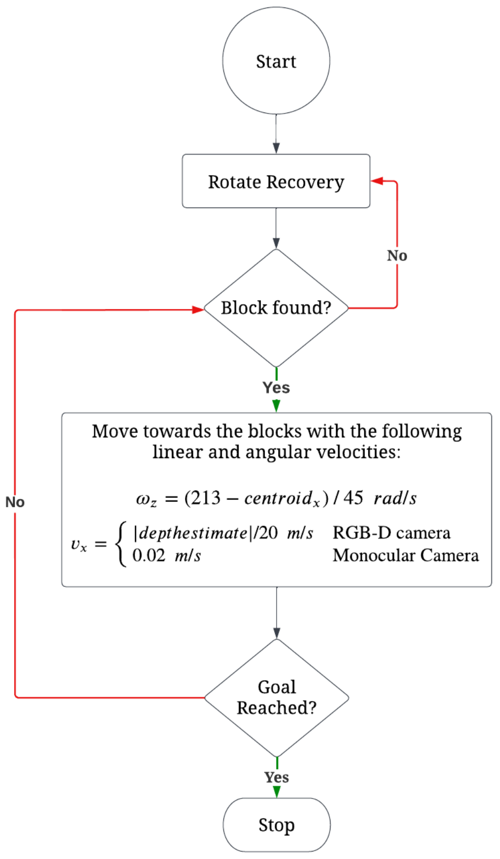
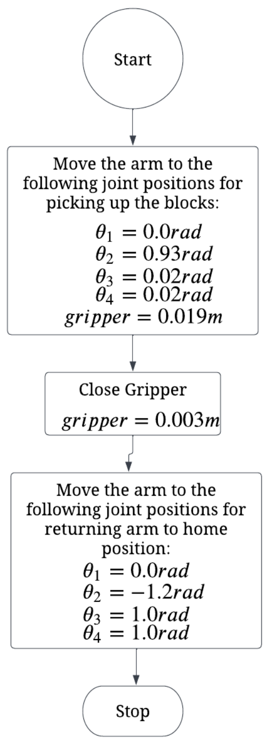

State 1
Navigate towards the material repository using the april_tags till threshold value is reached.
State 2
Navigate towards the blocks using object detection and depth estimation. Position robot in front of blocks at predetrmined pose
State 3
Pick up the block using the arm. The joint goal values are predetermined and the arm has a fixed motion.
State 4
Navigate towards the deposition zone using the april_tags till threshold value is reached.
State 5
Place the block using the arm. The joint goal values are predetermined and the arm has a fixed motion.

Figure 4.2 Finite-State Machine Describing the Robot's Control Architecture

Figure 4.3 illustrates the controller that governs the robot's actions in each of the states in Figure 4.2. In Figure 4.3(a), d denotes the distance between the reference frames of the robot's camera and the tag, respectively, in the x-direction, and the value of i is 0.3 for State 1 and 0.2 for State 4. In Figure 4.3(b), $centroid_x$ denotes the pixel coordinate corresponding to the centroid of the block along the horizontal axis. Since the input image to the object detection algorithm is resized to 416×416 pixels, the output for $centroid_x$ varies from 0 to 416 pixels, with the pixel value 213 representing the center of the image. Thus, the robot executes the states sequentially until it moves all blocks from the Material Repository to the Deposition Zone. If the robot is in State 1, 2, or 4 and moves too close to a wall, it moves backwards and, if it loses sight of the tag/block, it rotates until the tag/block reappears in its field of view (Figure 4.3(d)).


Start
Rotate Recovery
Tag Found?
No
Yes
Move towards the tag with the following linear and angular velocities:
$\omega_z = -(-i - orientation_z)\ \ rad/s$
$v_x = |\log(d)|/5\ \ m/s$
No
Robot within 0.5m from the tag
Yes
Stop


Start
Rotate Recovery
Block found?
No
Yes
Move towards the blocks with the following linear and angular velocities:
$\omega_z = (213 - centroid_x)\,/\,45\ \ rad/s$
$v_x = \begin{cases} |depthestimate|/20\ \ m/s & \text{RGB-D camera} \\ 0.02\ \ m/s & \text{Monocular Camera} \end{cases}$
No
Goal Reached?
Yes
Stop


Start
Move the arm to the following joint positions for picking up the blocks:
$\theta_1 = 0.0rad$
$\theta_2 = 0.93rad$
$\theta_3 = 0.02rad$
$\theta_4 = 0.02rad$
$gripper = 0.019m$
Close Gripper
$gripper = 0.003m$
Move the arm to the following joint positions for returning arm to home position:
$\theta_1 = 0.0rad$
$\theta_2 = -1.2rad$
$\theta_3 = 1.0rad$
$\theta_4 = 1.0rad$
Stop

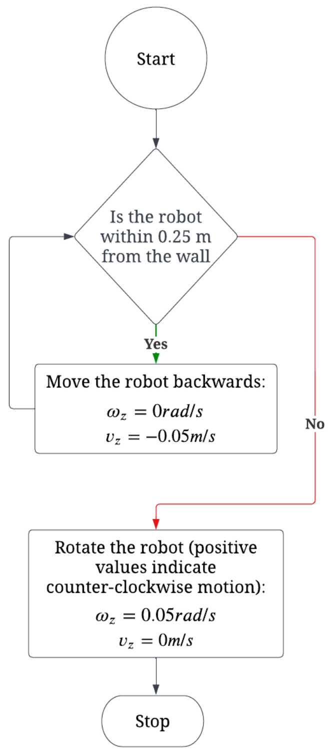

Start
Is the robot within 0.25 m from the wall
Yes
Move the robot backwards:
$\omega_z = 0rad/s$
$v_z = -0.05m/s$
No
Rotate the robot (positive values indicate counter-clockwise motion):
$\omega_z = 0.05rad/s$
$v_z = 0m/s$
Stop

Figure 4.3 Robot controller for each of the following states in Figure 4.2: (a, Top Left) Move robot base w.r.t AprilTags (State 1 and State 4), (b, Top Right) Move robot base w.r.t blocks (State 2), (c, Bottom Left) Move arm (State 3 and State 5), and (d, Bottom Right) Rotate recovery.

4.4 Simulation Setup – Assumptions

To validate the vision-based control approach outlined in this thesis, a test environment was set up in the Gazebo robot simulator (Koenig, 2004). The test world is a simplified approximation of a controlled construction environment with two zones: the Material Repository and the Material Deposition zone. The goal of the robot is to retrieve building blocks (one of the five classes described earlier, specified by the user) from the Material Repository, and transport them one at a time to the Material Deposition zone. The next section describes the Gazebo test world in greater detail.

The following assumptions are used in creating the simulation environment and defining the information available to the robot:

- The blocks that the robot needs to pick up are located on the ground, and so for the robot's onboard camera to be able to detect the blocks, it is placed at a lower position on the simulated robot than on the physical robot, as illustrated in Figure 4.4.
- To avoid collisions with walls in the simulated environment, the robot uses estimates of its distance to the walls from the depth estimation algorithm, based on images from its onboard camera and measurements from its onboard Lidar.

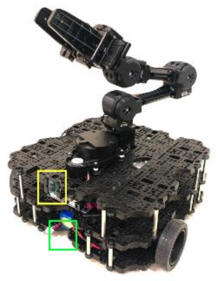


Figure 4.4 Position of the Camera on the Simulated Robot (Green Box) and on the Physical Robot (Yellow Box).

- The robot does not have a global localization mechanism and does not require prior mapping of the environment for it to execute the controller. Thus, the robot moves only based on its perception of the environment through images from its on-board camera, and it uses the AprilTags to help it navigate towards the different zones in the test environment.
- The controller does not depend on the dimensions of the robot. This makes the controller agent-agnostic, and it can thus potentially work on any mobile base with any arm, requiring minimal modifications to the control system. However, this approach has a downside: the robot could potentially collide with the objects in its environment, since the controller can only account for objects in the field of view of the camera. For instance, if the robot performs a 360° rotation close to a wall, the back part of the robot could potentially collide with the wall since the controller is unaware of the dimensions of the robot.
- The test environment is assumed to be free of obstacles; the only static obstacles in the environment are the walls of the simulated testbed.

- The test environment is assumed to have adequate lighting in order to eliminate issues pertaining to low lighting conditions such as problems with tag detections.
- The size of each AprilTag is $18 \times 18\ cm$. This ensures that the tags are visible from either end of the testbed, eliminating issues pertaining to tag detections.
- As mentioned in Section 4.2, the arm does not have sensor feedback, and so the robot’s success at picking up a block depends on the accuracy of the controller that drives its mobile base. However, adding feedback to the arm controller would improve the effectiveness of the robot at picking up blocks (e.g., its robustness to navigation errors) and would provide the robot with additional capabilities.

4.5 Simulation Setup – Description

The Gazebo simulation consists of the two zones mentioned earlier, the Material Repository and the Material Deposition zone. The testbed has the following dimensions: $4 \times 3 \times 0.3\ m$. Figure 4.5 illustrates a top-down view of the testbed in Gazebo, with two blocks located in the Repository zone. The red, green, and blue lines correspond to the x-axis, y-axis, and z-axis respectively, of the global coordinate frame.

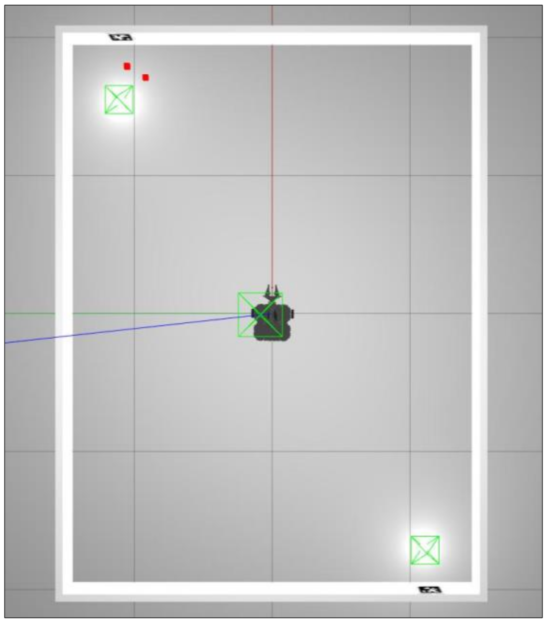


Figure 4.5 Top-Down View of the Testbed in Gazebo. The Red Blocks are Located in the Repository.

The repository and the deposition zones are positioned diagonally opposite to each other, and the robot is tasked with retrieving objects from the repository and transporting them to the deposition zone. Figure 4.6 depicts a closer view of the repository with different shape blocks and Table 4.1 outlines the dimensions of the blocks used in the simulation.

Table 4.1

Dimensions of the Blocks Used in Simulation

OBJECT	**DIMENSIONS**
Red and Green cubes	4 x 4 x 4 cm
Red and Green Cylinders	Radius: 2.5 cm; Length: 10 cm
Red Sphere	Radius: 2.5 cm

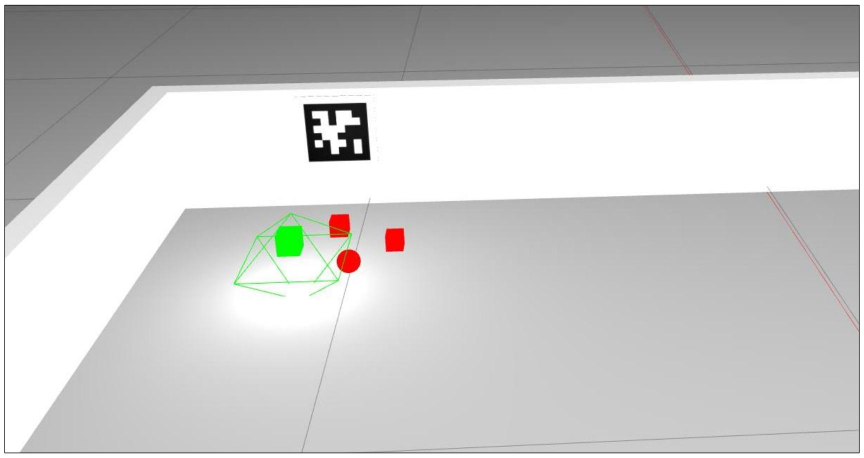


Figure 4.6 Closer View of the Repository Containing a Few Blocks in the Simulated Testbed.

4.6 Distributed Computing

To reduce the computational load, this thesis uses a distributed computing approach to running the simulations. The simulations leverage the power of the Robot Operating System (ROS) Melodic architecture to run the deep learning nodes on separate computers. Let $System_1$ and $System_2$ denote the two systems on which the nodes run. $System_1$ runs the ROS Master and the object detection node, while $System_2$ runs the depth estimation node. The systems are connected to each other with wired connections via a common router, which is in turn connected to the Arizona State University wired network, thus ensuring a bidirectional connection between the two systems. Table 4.2 highlights the specifications of both systems used to run the simulations.

Table 4.2

Specifications of the Systems Used to Run the Simulations

$System_1$	$System_2$
Intel Xeon E5-2623 processor	Intel i7-9750H processor
32 GB RAM	16 GB RAM
Nvidia Quadro M4000 8 GB GPU	Nvidia GTX 1660Ti 6 GB GPU

4.7 Experiment 1 – Simulation using Object Detection Algorithm and RGB-D Camera

To establish the baseline performance of the proposed vision-only control approach, the monocular camera was replaced by an RGB-D camera to obtain depth values directly, without estimating them from monocular images. The depth data obtained from the RGB-D camera was used in conjunction with the object detection algorithm to drive the robot to positions that facilitate block pickup. The next chapter discusses the results of the experiment.

4.8 Experiment 2 – Simulation Using Object Detection and Monocular Depth Estimation Algorithms

The second experiment was carried out using only the input from the robot's on-board monocular camera, together with the control approaches described earlier in the chapter. The next chapter discusses the results of the experiment.

CHAPTER 5

RESULTS AND DISCUSSION

5.1 Object Detection Algorithm

The object detection algorithm for identifying the building blocks was created by transfer learning on top of the YOLO v5 model. A set of 1087 images were collected from the Gazebo robot simulator by initially capturing a video of the objects of interest in the test bed, and individual frames were isolated to create the images required for training. The generated images were manually annotated using roboflow. Then data augmentation methods, namely changing the brightness and mirroring, were applied to increase the robustness of the model to variations in lighting and the orientations of the blocks. The size of the final dataset generated was 2777 images. The model was trained on the Google Colab platform on a Tesla P100 GPU. Although the model was trained for 500 epochs, there was no significant improvement in the model after epoch 294 (best model), and the training was terminated at epoch 394. The parameters of the dataset and the training routine used for transfer learning are summarized in Table 5.1.

Table 5.1

Training and dataset parameters

Parameter	**Value**
Number of Images	2777; Training: 2535, Test: 121, Validation: 121
Image Classes	5 (Red – cube, cylinder, sphere; Green – cube, cylinder)
Input Image size (pixels)	416×416

YOLO model	YOLOv5s
Batch size	16
Number of Epochs	500

The results of transfer learning are shown in the following plots: Confusion matrix (Singh 2021) (Figure 5.1), F1 curve (Kulkarni, 2020) (Figure 5.2), and Precision-Recall curve (Ozenne, 2015) (Figure 5.3).

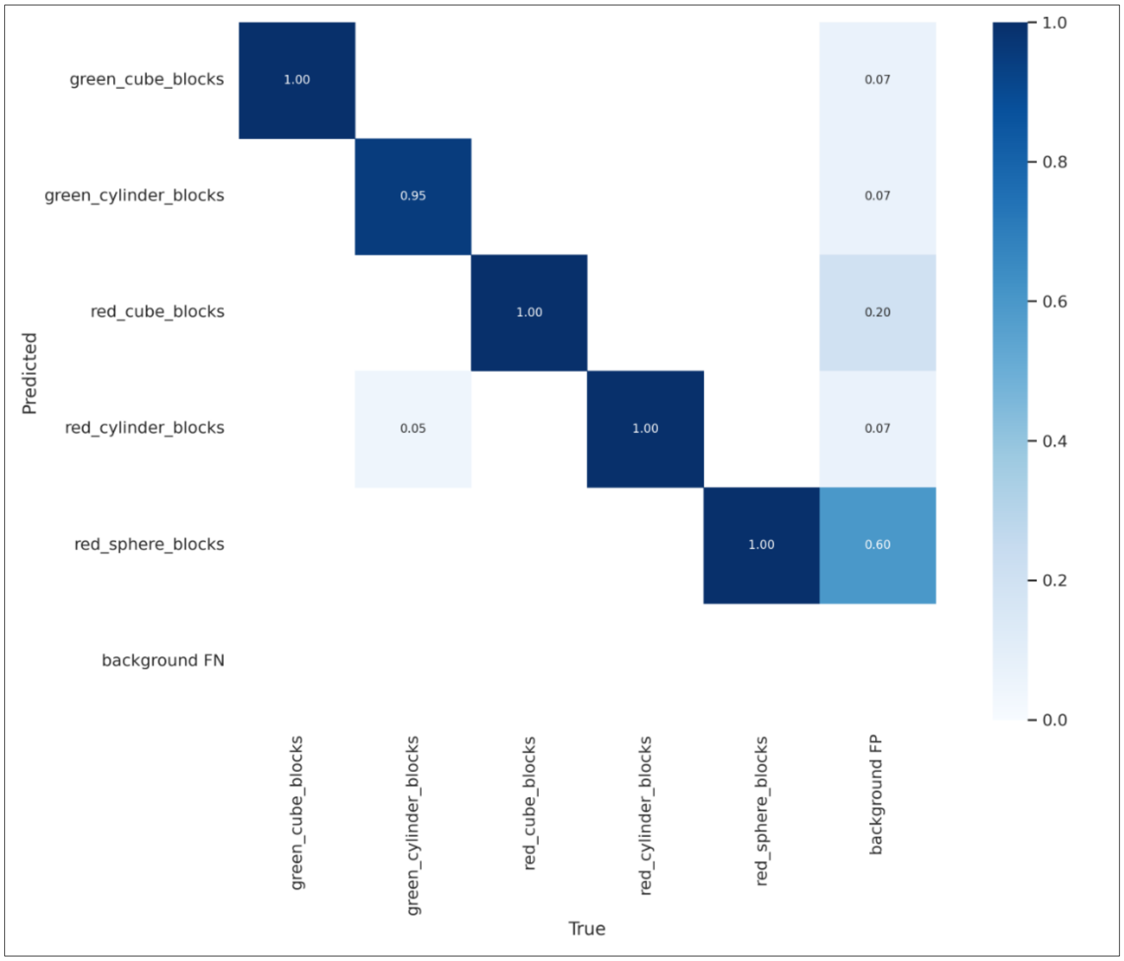


Figure 5.1 Confusion Matrix

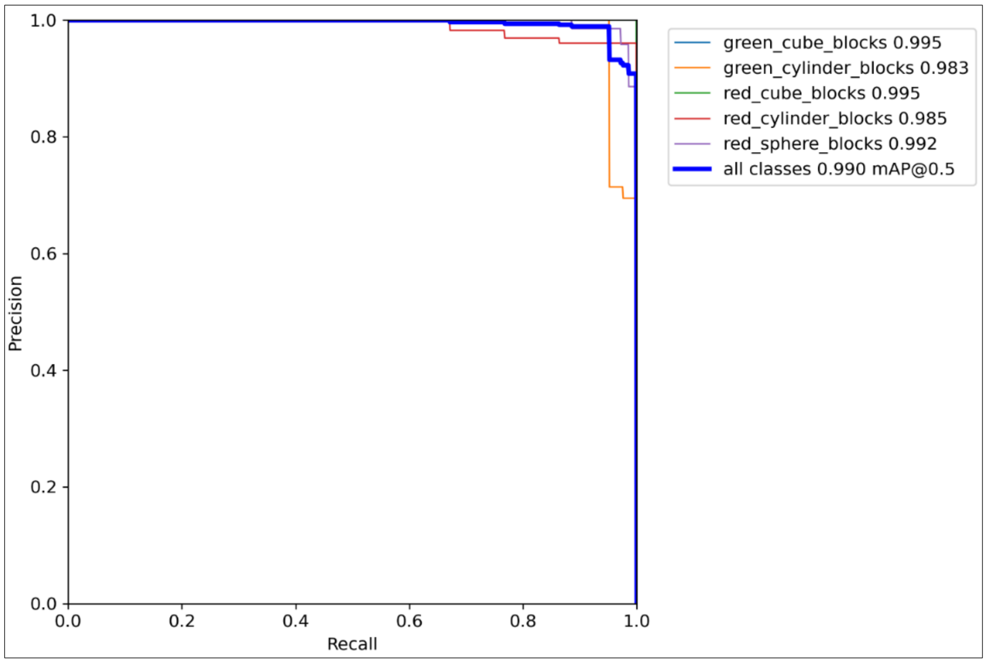


Figure 5.2 Precision-Recall Curve

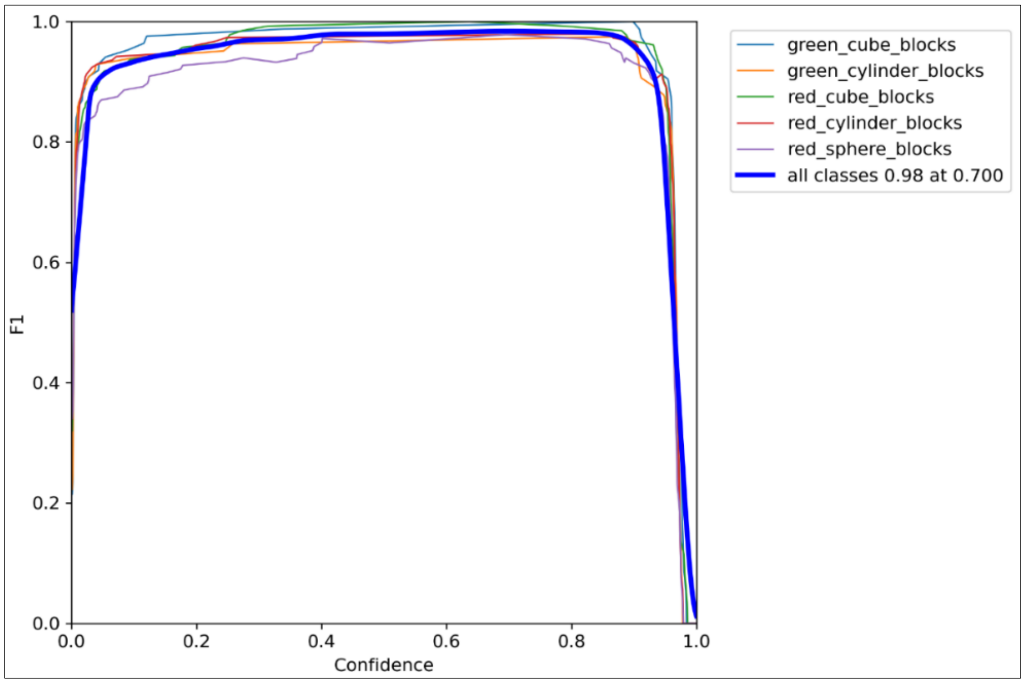


Figure 5.3 F1 Curve

From the plots above, it is evident that the model fits the data very well, since it has a mean average precision (mAP) of 0.99 and an average F1 score of 0.98 at a confidence level of 0.7. Some sample predictions from the model on the test dataset are shown in Figure 5.4. Additional results on the performance of the algorithm are included in Appendix B, and the reader is encouraged to refer to that section for further information.

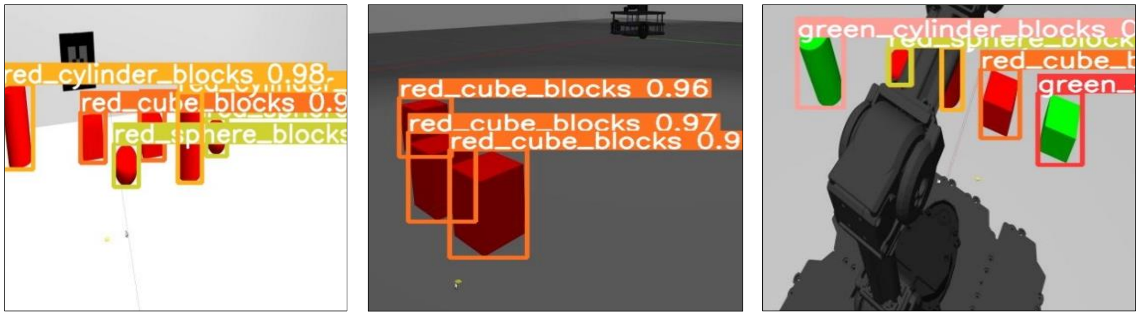


Figure 5.4 Sample Predictions of the Model for Images From the Test Dataset. The Numeric Values Indicate the Confidence Levels for Each Identified Object.

5.2 Simulation 1 – Using RGB-D Camera

The first simulation uses an RGB-D camera (Section 4.7) to set a performance benchmark for the controller. The robot's goal is to retrieve one block (red cube) at a time from the Material Repository and transport it to the Deposition Zone. Once the robot picks up a block from the repository, another block is manually replaced in the Material Repository at a different location with respect to the previous block. Only one block is present at a time in the Material Repository to avoid sudden variations in the value of $centroid_x$ that arise when the object detection algorithm identifies multiple blocks in a single image. Section 5.2.1 discusses this in further detail.

A *Trial* is defined as a single iteration in the simulation where the robot navigates to the Material Repository, identifies a block, picks it up with its manipulator, navigates to the Deposition Zone, and places the block down. A *Trial* is successful (*Success*) if the robot correctly executes each of the actions in this sequence. If the robot fails to execute any action in a trial successfully, then the *Trial* is a failed run (*Fail*). Since the control of the arm is open-loop, the controller cannot determine from sensor feedback whether the robot has indeed picked up a block. Thus, the robot still performs a complete *Trial* even in the case of a *Fail*; however, in such a case the robot does not transport the block to the Deposition Zone (it arrives there "empty-handed").

A human observer determines whether the outcome of a *Trial* is a *Success* or a *Fail*. The controller was tested for 50 *Trial*s, and the outcome of each trial is plotted in the graph shown in Figure 5.5. The value 1 on the y-axis corresponds to *Success* and the value 0 corresponds to *Fail*. This implies that the green regions of the graph correspond to successful *Trial*s and the white regions correspond to failed attempts.

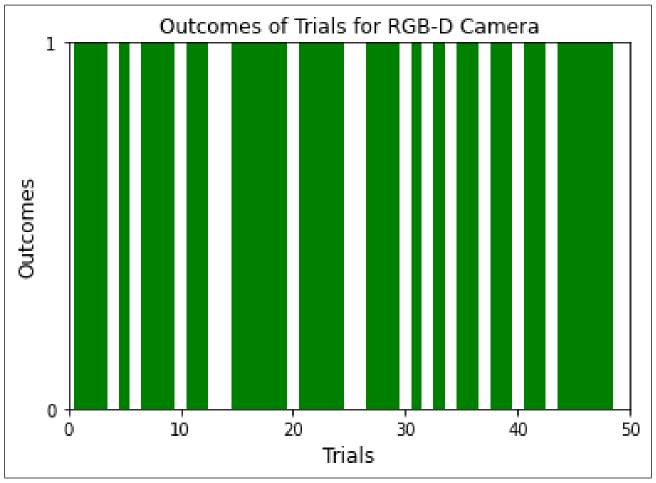


Figure 5.5 Outcomes of 50 *Trial*s for the Simulation Using RGB-D Camera

Since 34 out of 50 *Trials* were successful, the controller has a success rate of 68%. Figure 5.6 shows snapshots of the robot picking up a block during several successful *Trial*s for Simulation 1.

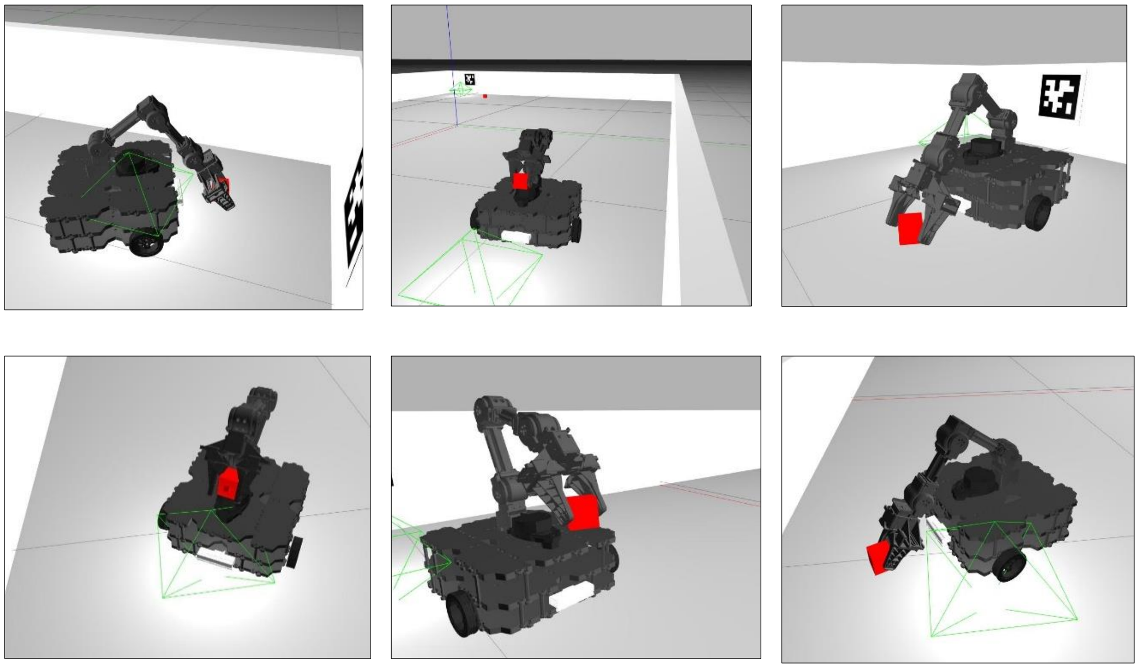


Figure 5.6 Instances of Successful Block Pickup in Simulation 1

5.2.1 Failure Analysis

It is imperative to mention the different cases under which the controller *Fails* in order to understand the overall performance of the system better. Figure 5.7 illustrates a few examples where the controller *Fails*. Based on observations of the outcomes for of 25 *Trials*, three major factors have been identified which cause a *Trial* to *Fail*. They are: **Failure Mode 1:** Failure during navigation towards Material Repository (State 1, Section

4.3), **Failure Mode 2:** Errors in the object detection algorithm, and **Failure Mode 3:** Sensor errors. The subsequent paragraphs discuss the Failure Modes in greater detail.

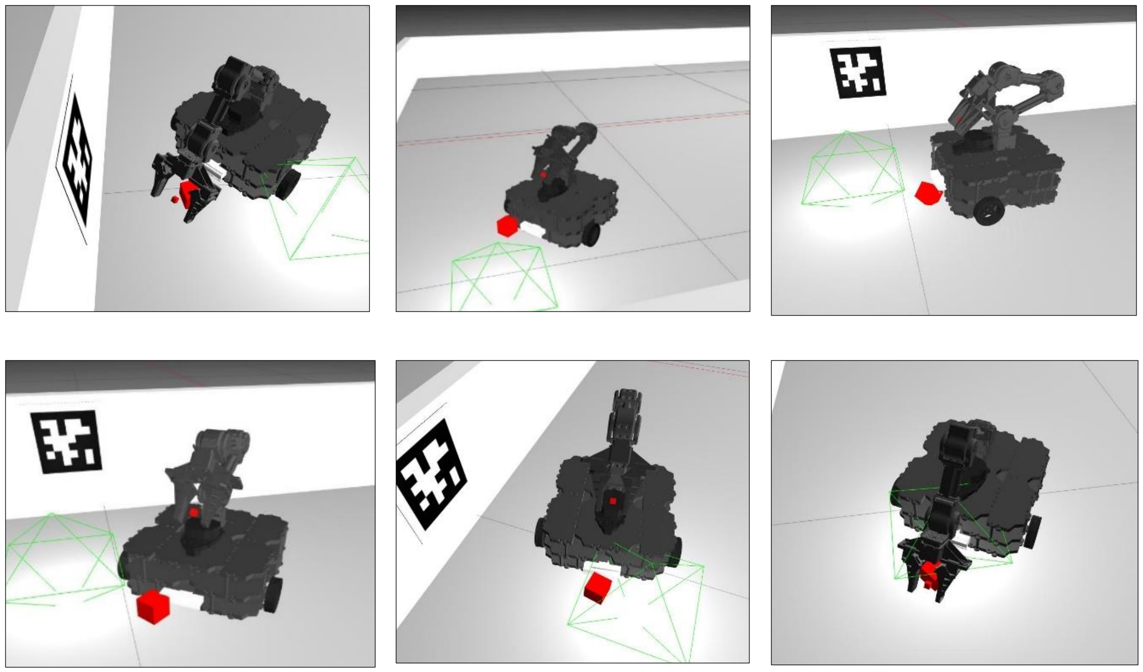


Figure 5.7 Instances of Failed Block Pickup in Simulation 1

Failure Mode 1: The tags corresponding to the Material Repository and the Deposition zones are placed almost diagonally opposite to each other. Since the shortest path between two given points is a straight line, the controller is set up to traverse a straight line while navigating between the two zones. However, since there are an infinite number of parallel lines with the same slope, the robot could traverse along any parallel line, which could cause the controller to potentially *Fail*.

To understand this further, consider Figure 5.7. The blue region represents the approximate operating region where the controller executes State 1 and State 4 as intended.

The red arrow represents the direction along which the robot and the tags are aligned. The maroon lines represent the parallel lines along which the robot could potentially traverse. When the robot loses sight of a tag, it performs a Rotate Recovery behavior (Figure 4.3) until the robot identifies the tags. At this point, the robot continues to move along another parallel line with the same slope determined by the robot controllers associated with State 1 and State 4 (Section 4.3).

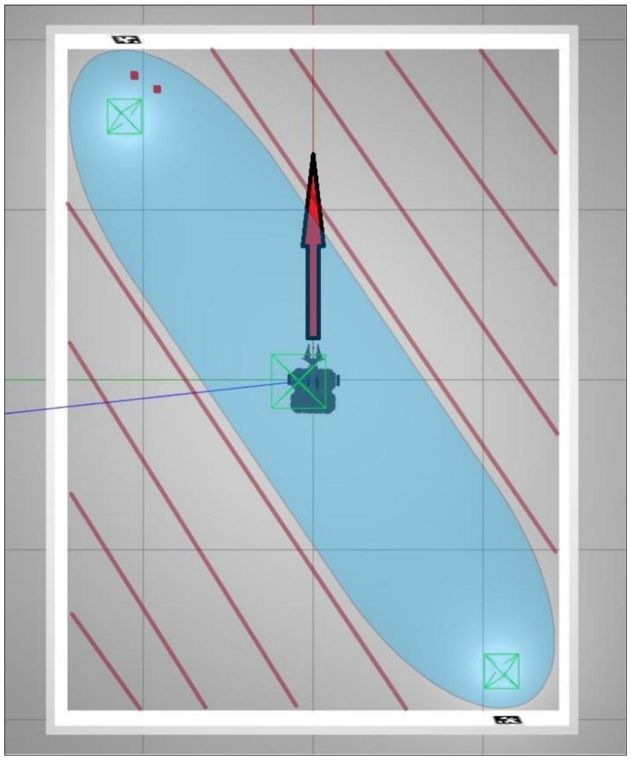


Figure 5.8 Illustration for Failure Mode 1

Therefore, when the robot picks a line below the blue operating region, it could potentially traverse along a path like the purple trajectory in Figure 5.9. Similarly, when the robot picks a line above the blue operating region, it could display a behavior such as the yellow trajectory shown in Figure 5.9. Thus, in the unlikely case that the robot steers far off from the blue operating region, it could potentially strike the walls of the test bed and not exhibit the desired behaviors. A different controller for navigating between the tags could potentially resolve this problem.

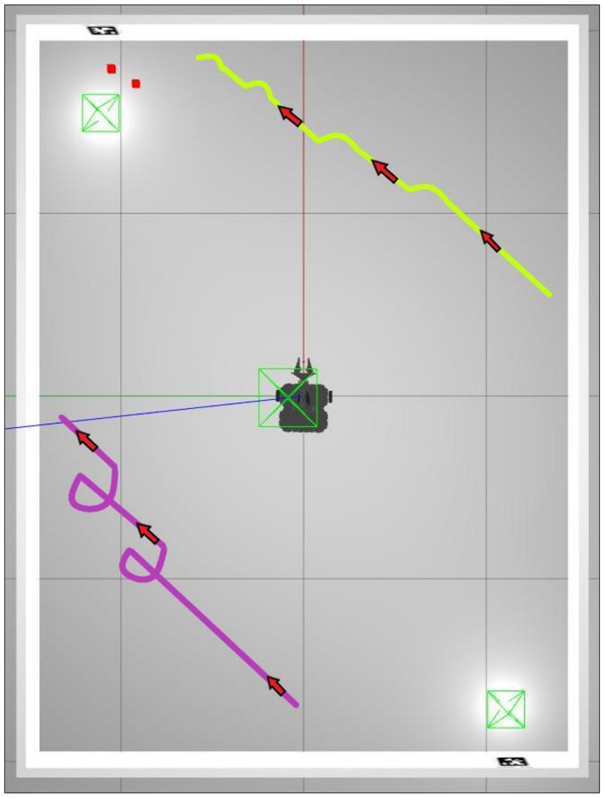


Figure 5.9 Sample Robot Paths for Failure Mode 1. Red Arrows Indicate Direction of Travel.

Failure Mode 2 is caused by erroneous outputs from the object detection algorithm. Figure 5.10 illustrates some examples of such erroneous outputs. Erroneous outputs cause the value of $centroid_x$ to fluctuate suddenly, which then causes sudden variations in the angular velocity commands. Thus, the robot could suddenly steer off course and miss the block pickup, which causes the *Trial* to *Fail*. This behavior is especially pronounced when multiple blocks are in the Repository Zone. Since the object detection algorithm cannot uniquely identify instances of objects within a given image, the order in which the objects are detected varies with each frame and thus causes the output value of $centroid_x$ to fluctuate. Using an instance segmentation algorithm instead of the object detection algorithm could solve this problem.

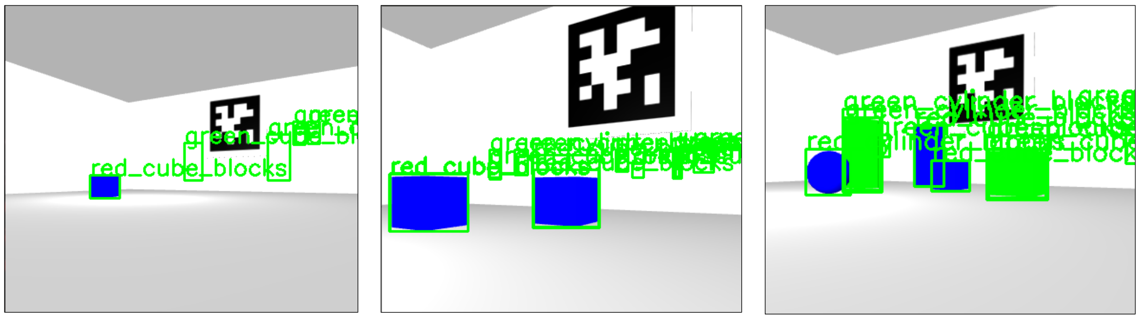


Figure 5.10 Sample Erroneous Outputs of the Object Detection Algorithm in the Simulations

Failure Mode 3 occurs due to errors and uncertainty associated with the depth estimates from the RGB-D camera. These depend upon the type of camera used and its characteristics. Based on observations, this type of failure was relatively rare and could be easily eliminated by using a more accurate RGB-D camera.

5.3 Simulation 2 – Using Monocular Camera

The second simulation replaces the RGB-D camera with a monocular camera as discussed in Chapter 4. The definitions for *Trial*, *Success* and *Fail* are as defined in Simulation 1 (Section 5.2). The controller was tested for 50 Trial’s and the outcomes have been plotted in the Graph as shown in Figure 5.11. The value 1 on the y-axis corrsponds to *Success* and the value 0 corresponds to *Fail*. This implies that the green regions of the graph correspond to successful *Trial*'s and the white regions correspond to failed attempts.

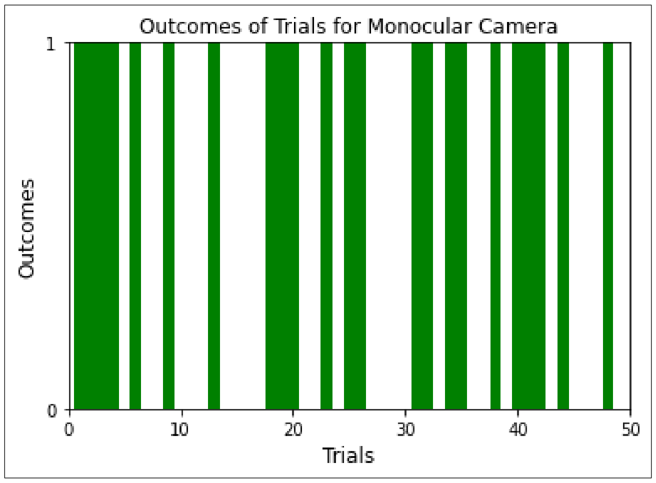
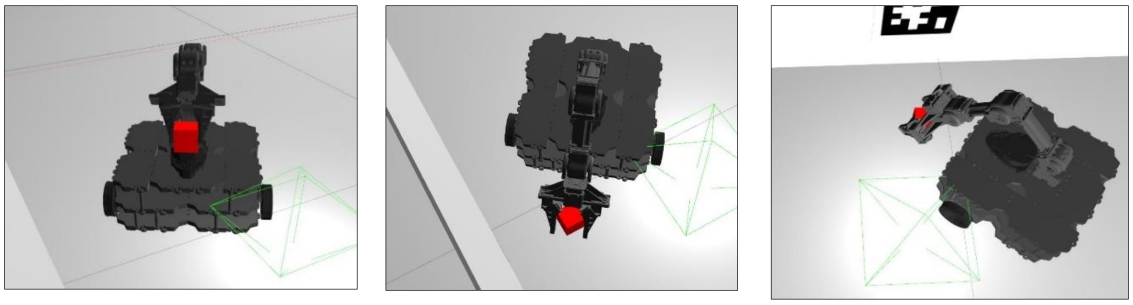


Figure 5.11 Outcomes of 50 *Trial*s for the Simulation Using Monocular Camera

Since 23 out of 50 *Trial*s were successful, the controller has a success rate of 46%. Figure 5.12 shows snapshots of the robot picking up a block during several successful *Trial*s for Simulation 2.



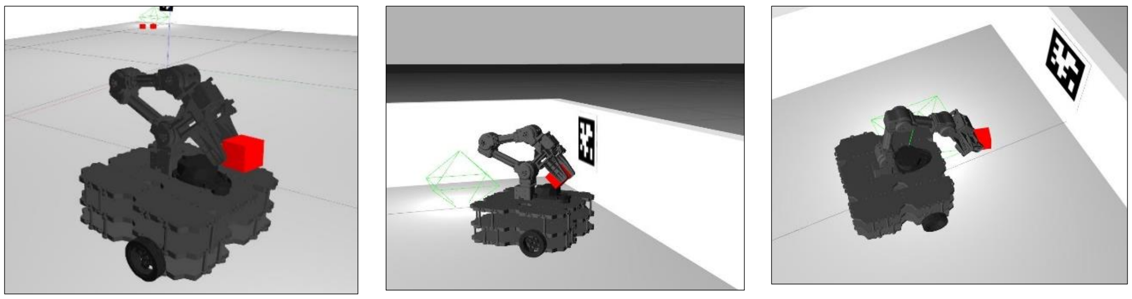
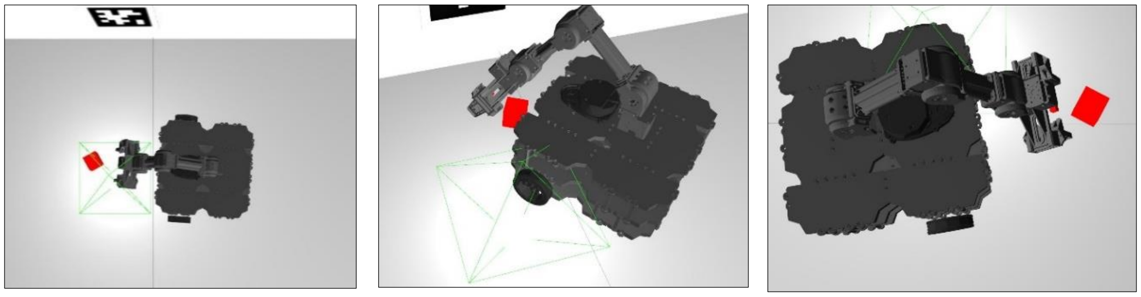


Figure 5.12 Instances of Successful Block Pickup in Simulation 2

5.3.1 Failure Analysis

The failure analysis for Simulation 2 is similar to that for Simulation 1. Figure 5.13 illustrates a few examples where the controller *Fails*. The following modes of failure were observed: **Failure Mode 1:** Failure during navigation towards Material Repository (State 1, Section 4.3), **Failure Mode 2:** Errors in the object detection algorithm, and **Failure Mode 3:** Failure due to inconsistent depth estimates from the depth estimation algorithm. **Failure Modes 1** and **3** have been discussed in detail in the previous section (Section 5.2.1). The subsequent paragraphs elaborate on **Failure Mode 3**.



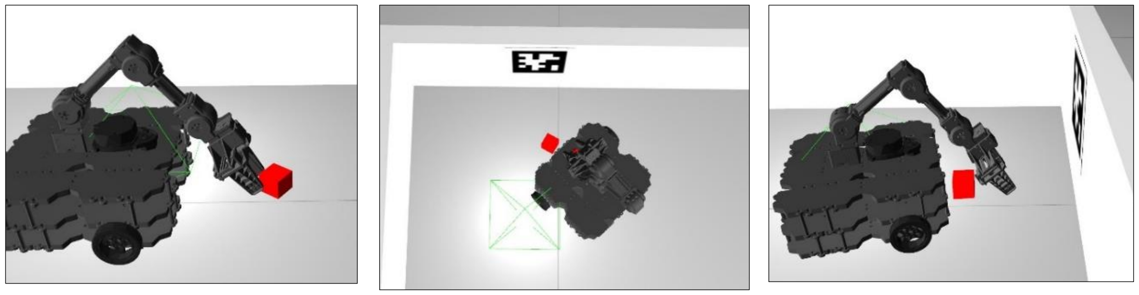
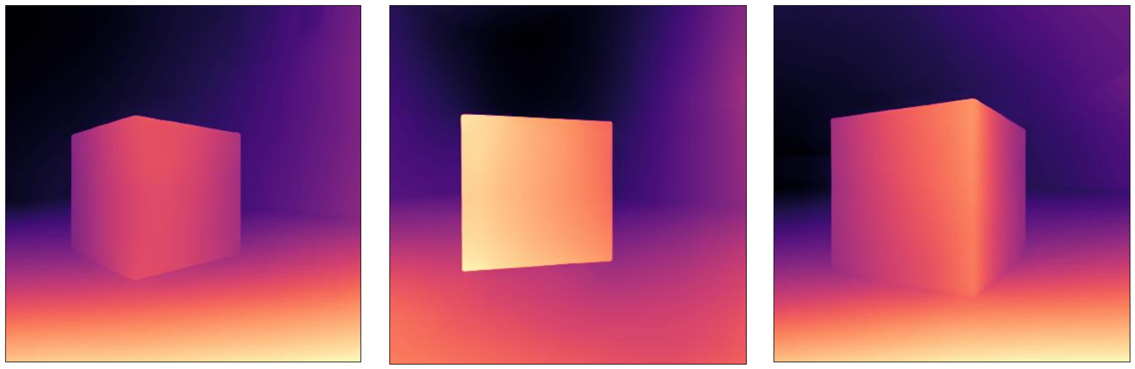


Figure 5.13 Instances of Failed Block Pickup in Simulation 2

Failure Mode 3 in Simulation 2 occurs due to the variations in the depth estimates from the depth estimation algorithm. Figure 5.14 illustrates variations in the depth map generated by the algorithm for an input image during the robot's transition from State 2 to State 3 (Section 4.3). Brighter points in the image correspond to the regions closer to the camera, according to the algorithm, and darker points correspond to regions farther away from the camera. Such variations in the depth estimates cause the controller to misjudge the distance of the block from the robot and result in a *Failed Trial*.



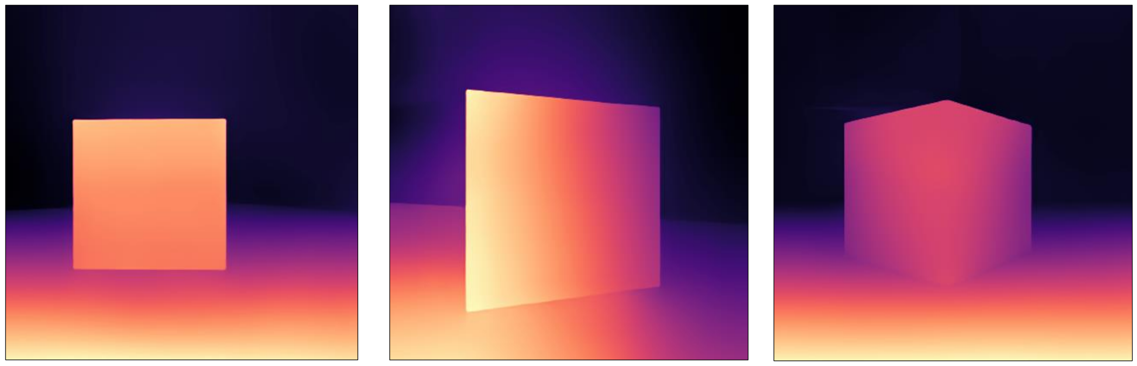


Figure 5.14 Depth Map for Different Runs Before Block Pickup

However, a closer examination of the failed runs in Simulation 2 and Figure 5.14 provides insight into the margin by which the pickup fails in most *Trials*. Figure 5.15 illustrates the region around the block where the gripper attempts a pickup in the case of a *Fail*. This region has an approximate margin of 2 inches on all sides measured with respect to the vertical faces of the block. Thus, the lower success rate of the controller can largely be attributed to the slight variations in the estimates of the depth estimation algorithm, which causes the pickup to fail within a tolerance of approximately ± 2 inches on all sides with respect to the block, which is a very low margin of error. Transfer learning the depth estimation model on a custom dataset obtained from the simulations could improve the stability of the depth estimates, thus increasing the success rate of the controller.

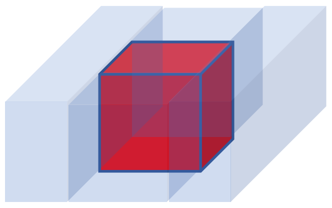


Figure 5.15 Typical Failure Envelope around a block in Simulation 2

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this thesis, we developed a novel vision-based controller that combines existing techniques in deep-learning-based object detection and depth estimation using a monocular camera. The controller is designed to be used for material pick-and-place tasks in construction applications. Sensor information from a single monocular camera is the only input that the controller requires. There is no global localization mechanism, and the controller does not require prior mapping of the environment. Two simulations of 50 trials each were performed. The first simulation used an RGB-D camera to benchmark the controller's performance, and the success rate was calculated to be 68% over the 50 trials. In the second simulation, the RGB-D camera was replaced with a monocular camera, and the success rate, in this case, was determined to be 46% over the 50 trials.

To better understand the controller's performance, a thorough analysis of the cases under which the controller fails was performed. Based on the analysis, the lower success rate observed in the second simulation was attributed to the unstable depth estimates from the depth estimation network. Thus, improving the stability of the depth estimates from the network could potentially increase the success rate of the robot in Simulation 2.

6.2 Future Work

Some possible extensions of the work in this thesis are as follows:

- Train the depth estimation model using a custom dataset derived from the simulation environment to obtain more stable and consistent depth estimates, which could improve the success rate of the robot.
- Replace object detection with instance segmentation to uniquely identify different instances of objects in a single camera frame, such as blocks of different colors and shapes, which could potentially avoid Failure Mode 2 (Section 5.2.1 and Section 5.3.1).
- Train the object detection model to identify more classes of objects.
- Add force sensing or vision-based feedback to the arm, to enable the use of closed-loop controllers that can compensate for errors during block pickup and deposition, thus potentially increasing the success rate of the robot.
- Extend the proposed control approach to develop robot controllers for collective construction tasks performed by multiple robots, incorporating rules for collision avoidance and coordination among the robots.

REFERENCES

Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2012). *Robotics: Modelling, Planning and Control*. Springer London.

Lynch, K. M., & Park, F. C. (2019). *Modern Robotics: Mechanics, planning, and Control*. Cambridge University Press.

Tzafestas, S. G. (2014). *Introduction to Mobile Robot Control*. Elsevier.

Sandakalum, T., & Ang, M. H. (2022). Motion Planning for Mobile Manipulators—A Systematic Review. *Machines*, *10*(2), 97. https://doi.org/10.3390/machines10020097

Szeliski, R. (2011). *Computer vision algorithms and applications*. Springer.

Ballard, D. H., & Brown, C. M. (1981). *Computer Vision*. Prentice Hall.

Huang, T.S. (1996). Computer Vision: Evolution and Promise. *CERN School of Computing*, 21-25. DOI: 10.5170/CERN-1996-008.21

Zhao, C., Sun, Q., Zhang, C., Tang, Y., & Qian, F. (2020). Monocular depth estimation based on deep learning: An overview. *Sci. China Technol. Sci.* 63, 1612–1627. https://doi.org/10.1007/s11431-020-1582-8

Eigen, D., Puhrsch, C., & Fergus, R. (2014). Depth map prediction from a single image using a multi-scale deep network. *Proceedings of the 27th International Conference on Neural Information Processing Systems*, (2), 2366-2374. https://dl.acm.org/doi/10.5555/2969033.2969091

Luo, X., Huang, J.-B., Szeliski, R., Matzen, K., & Kopf, J. (2020). Consistent video depth estimation. *ACM Transactions on Graphics*, *39*(4), 71:1-71:13 https://dl.acm.org/doi/abs/10.1145/3386569.3392377

Zhao, Z. Q., Zheng, P., Xu, S. T., & Wu, X. (2019). Object Detection with Deep Learning: A Review, *IEEE Transactions on Neural Networks and Learning Systems*, *30*(11), 3212-3232. doi:10.1109/TNNLS.2018.2876865.

Aggarwal, C. C. (2018). *Neural Networks and Deep Learning*. Springer Cham.

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. *IEEE International Conference on Computer Vision (ICCV)*, 2980-2988.

Cortes, C., & Vapnik, V. (1995). Support vector machine. *Machine Learning*, *20*(3), 273–297. https://doi.org/10.1007/BF00994018

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, *60*(2), 91–110. https://doi.org/10.1023/B:VISI.0000029664.99615.94

Mertan, A., Duff, D. J., & Unal, G. (2022). Single image depth estimation: An overview. *Digital Signal Processing, 123*. https://doi.org/10.1016/j.dsp.2022.103441.

Joglekar, A., Joshi, D., Khemani, R., Nair, S., & Sahare, S. (2011). Depth Estimation Using Monocular Camera. *International Journal of Computer Science and Information Technologies*. *2*(4), 1758-1763.

Ranftl, R., Lasinger, K., Hafner, D., Schindler, K., & Koltun, V. (2022). Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-Shot Cross-Dataset Transfer. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, *44*(3), 1623-1637. Doi: 10.1109/TPAMI.2020.3019967

Scharstein, D., & Szeliski, R. (2003). High-accuracy stereo depth maps using structured light. *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. Proceedings.*, I-I, Doi: 10.1109/CVPR.2003.1211354.

Zhou, B., Krähenbühl, P., & Koltun, V. (2019). Does computer vision matter for action? *Science Robotics, 4(30)*. Doi: 10.1126/scirobotics.aaw6661

Geiger, A., Lenz, P., & Urtasun, R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. *2012 IEEE Conference on Computer Vision and Pattern Recognition.* 3354–3361. Doi: 10.1109/CVPR.2012.6248074.

Silberman, N., Hoiem, D., Kohli, P., &Fergus, R. (2012). Indoor Segmentation and Support Inference from RGBD Images. In: Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y., Schmid, C. (eds) Computer Vision – ECCV 2012. ECCV 2012. Lecture Notes in Computer Science, *7576*. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-33715-4_54

Saxena, A., Schulte, J., & Ng, A. Y. (2007). Depth estimation using monocular and stereo cues. *Proceedings of the 20th international joint conference on Artificial intelligence (IJCAI'07).* 2197-2203.

Ullman, S. (1979). The interpretation of structure from motion. *Proc. R. Soc. Lond. B.203*, 405–426. http://doi.org/10.1098/rspb.1979.0006

Mitchell, T. M. (1997). *Machine Learning*. McGraw Hill.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 779-788. Doi: 10.1109/CVPR.2016.91.

Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, Faster, Stronger. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6517-6525.

Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. *ArXiv*, abs/1804.02767.

Bochkovskiy, A., Wang, C., & Liao, H.M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. *ArXiv*, abs/2004.10934

Jocher, Glenn, et al. (2020) Ultralytics/Yolov5: V3.1 - Bug Fixes and Performance Improvements. *Zenodo*, https://doi.org/10.5281/zenodo.4154370.

Pan, S. J., & Yang, Q. (2010). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, *22*(10), 1345-1359. Doi: 10.1109/TKDE.2009.191.

Štibinger, P., et al., (2021). Mobile Manipulator for Autonomous Localization, Grasping and Precise Placement of Construction Material in a Semi-Structured Environment. IEEE Robotics and Automation Letters, *6*(2), 2595-2602. Doi: 10.1109/LRA.2021.3061377.

Basiri, M., Gonçalves, J., Rosa, J., Vale, A., & Lima, P. (2021). An autonomous mobile manipulator to build outdoor structures consisting of heterogeneous brick patterns. SN Appl. Sci. 3, 558. https://doi.org/10.1007/s42452-021-04506-7

Lussi, M., *et al.*, (2018) Accurate and Adaptive in Situ Fabrication of an Undulated Wall Using an on-Board Visual Sensing System. *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 3532-3539. Doi: 10.1109/ICRA.2018.8460480.

Asadi, K., Haritsa, V.R., Han, K.K., & Ore, J. (2021). Automated Object Manipulation Using Vision-Based Mobile Robotic System for Construction Applications. *Journal of Computing in Civil Engineering, 35*, 04020058.

Wang, C., Zhang, Q., Tian, Q., Li, S., Wang, X., Lane, D., Petillot, Y., & Wang, S. (2020). Learning Mobile Manipulation through Deep Reinforcement Learning. *Sensors*, *20*(3), 939. MDPI AG. http://dx.doi.org/10.3390/s20030939

Iriondo, A., Lazkano, E., Susperregi, L., Urain, J., Fernandez, A., & Molina, J. (2019). Pick and Place Operations in Logistics Using a Mobile Manipulator Controlled with Deep Reinforcement Learning. *Applied Sciences*, *9*(2), 348. MDPI AG. http://dx.doi.org/10.3390/app9020348

Delgado, J. M. D., Oyedele, L., Ajayi, A., Akanbi, L., Akinade, O., Bilal, M., & Owolabi, H. (2019). Robotics and automated systems in construction: Understanding industry-specific challenges for adoption, *Journal of Building Engineering, 26.* https://doi.org/10.1016/j.jobe.2019.100868.

Jung, D. H., Park, J., & Schwartz, M. (2014). Towards on-site autonomous robotic floor tiling of mosaics. *14th International Conference on Control, Automation and Systems (ICCAS 2014)*, 59-63. Doi: 10.1109/ICCAS.2014.6987959.

Feng, C., Xiao, Y., Willette, A., McGee, W., & Kamat, V. R. (2014). Towards Autonomous Robotic In-Situ Assembly on Unstructured Construction Sites Using Monocular Vision. *Proceedings of the 31st International Symposium on Automation and Robotics in Construction and Mining (ISARC)*, 163–170. doi:10.22260/ISARC2014/0022

Feng, C., Xiao, Y., Willette, A., McGee, W., & Kamat, V. R. (2015). Vision guided autonomous robotic assembly and as-built scanning on unstructured construction sites. *Automation in Construction, 59,* 128–138. doi:10.1016/j.autcon.2015.06.002

Asadi, K., Ramshankar, H., Pullagurla, H., Bhandare, A., Shanbhag, S., Mehta, P., Kundu, S., Han, K., Lobaton, E., & Wu, T. (2018). Vision-based integrated mobile robotic system for real-time applications in construction. *Automation in Construction, 96*, 470–482. doi:10.1016/j.autcon.2018.10.009

Cilia, J. (2019). The Construction Labor Shortage: Will Developers Deploy Robotics? Forbes. https://bit.ly/3QD8aNn

Garg, S., Sunderhauf, N., Dayoub, F., Morrison, D., Cosgun, A., Carneiro, G., Wu, Q., Chin, T., Reid, I.D., Gould, S., Corke, P., & Milford, M. (2020). Semantics for Robotic Mapping, Perception and Interaction: A Survey. *Found. Trends Robotics, 8*, 1-224.

Petersen, K. H., Napp, N., Stuart-Smith, R., Rus, D., & Kovac, M. (2019). A review of collective robotic construction. *Science Robotics*, *4*(28), eaau8479. doi:10.1126/scirobotics.aau8479

Werfel J., (2012). "Collective Construction with Robot Swarms", Morphogenetic Engineering: Toward Programmable Complex Systems, *Springer Berlin Heidelberg*, pp. 115-140, ISBN: 978-3-642-33902-8.

Petersen, K., & Nagpal, R. (2017). Complex Design by Simple Robots: A Collective Embodied Intelligence Approach to Construction. *Archit. Design, 87,* 44-49. https://doi.org/10.1002/ad.2194

Florida, R., Rodríguez-Pose, A., & Storper, M. (2021). Cities in a post-COVID world. *Urban Studies*. https://doi.org/10.1177/00420980211018072

Robotis. (n.d.). *Turtlebot3 Emanual*. Robotis Co., Ltd. https://bit.ly/3NdEhAj

D. Malyuta. (2017). Navigation, Control and Mission Logic for Quadrotor Full-cycle Autonomy. *Master thesis*, Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, CA 91109, USA. https://doi.org/10.3929/ethz-b-000248154.

Shirsat, A., Elamvazhuthi, K., & Berman, S. (2020). Multi-Robot Target Search using Probabilistic Consensus on Discrete Markov Chains. *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR),* 108–115. doi:10.1109/SSRR50563.2020.9292589

Wang, J., & Olson, E., (2016). AprilTag 2: Efficient and robust fiducial detection. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4193–4198. https://doi.org/10.1109/IROS.2016.7759617

Coleman, D., Șucan, J. A., Chitta, S., & Correll, N., (2014). Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study. Journal of Software Engineering for Robotics, 5(1), 3-16. Doi: 10.6092/JOSER_2014_05_01_p3.

Șucan, I. A., Moll, M., & Kavraki, L. E., (2012). The Open Motion Planning Library. *IEEE Robotics & Automation Magazine, 19*(4), 72–82. https://ompl.kavrakilab.org

Koenig, N., & Howard, A., (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, *3*, pp. 2149-2154. Doi: 10.1109/IROS.2004.1389727.

Tsai, C. Y., Chou, Y. S., Wong, C., C., Lai, Y. C., & Huang, C., C. (2020). Visually Guided Picking Control of an Omnidirectional Mobile Manipulator Based on End-to-End Multi-Task Imitation Learning. *IEEE Access*, *8*, pp.1882-1891, doi:10.1109/ACCESS.2019.2962335.

Lu, Q., Fricke, G.M., Ericksen, J.C. et al. (2020). Swarm Foraging Review: Closing the Gap Between Proof and Practice. Curr Robot Rep, *1*, 215–225. https://doi.org/10.1007/s43154-020-00018-1

Paszke, A., Chaurasia, A., Kim, S., & Culurciello, E. (2016). ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation. *ArXiv, abs/1606.02147*.

Mur-Artal, R., Montiel, J. M. M., & Tardos, J. D., (2015). ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *Trans. Rob. 31*(5), 1147–1163. https://doi.org/10.1109/TRO.2015.2463671

Labbé, M., Michaud, F., (2019). RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *J Field Robotics*, *35*, 416– 446. https://doi.org/10.1002/rob.21831

Asadi, K., Chen. P., Han, K., Wu, T., & Lobaton, E. (2019). LNSNet: Lightweight Navigable Space Segmentation for Autonomous Robots on Construction Sites. *Data*. *4*(1), 40. https://doi.org/10.3390/data4010040

Ratliff, N., Zucker, M., Bagnell, J. A., & Srinivasa, S. (2009). "CHOMP: Gradient optimization techniques for efficient motion planning". *IEEE International Conference on Robotics and Automation,* pp. 489-494. Doi: 10.1109/ROBOT.2009.5152817.

Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., & Schaal, S. (2011). STOMP: Stochastic trajectory optimization for motion planning. *IEEE International Conference on Robotics and Automation*, pp. 4569-4574, Doi: 10.1109/ICRA.2011.5980280.

Kulkarni, A., Chong, D., & Batarseh, F. A. (2020). 5 - Foundations of data imbalance and solutions for a data democracy. *Data Democracy,* pp. 83–106. doi:10.1016/B978-0-12-818366-3.00005-8

Singh, P., Singh, N., Singh, K. K., & Singh, A. (2021). Chapter 5 - Diagnosing of disease using machine learning. *Machine Learning and the Internet of Medical Things in Healthcare,* pp. 89–111. doi:10.1016/B978-0-12-821229-5.00003-3

Shirsat, A., Mishra, S., Zhang, W., & Berman, S. (2022). Probabilistic Consensus on Feature Distribution for Multi-Robot Systems With Markovian Exploration Dynamics. *IEEE Robotics and Automation Letters*, *7*(3), 6407–6414. doi:10.1109/LRA.2022.3171905

Ozenne, B., Subtil, F., & Maucort-Boulch, D. (2015). The precision–recall curve overcame the optimism of the receiver operating characteristic curve in rare diseases. *Journal of Clinical Epidemiology, 68*(8), 855–859. doi:10.1016/j.jclinepi.2015.02.010

MBZIRC. (n.d.). Www.mbzirc.com. Retrieved August 20, 2022, from https://www.mbzirc.com/

APPENDIX A

YOLO v5 SYSTEM ARCHITECTURE

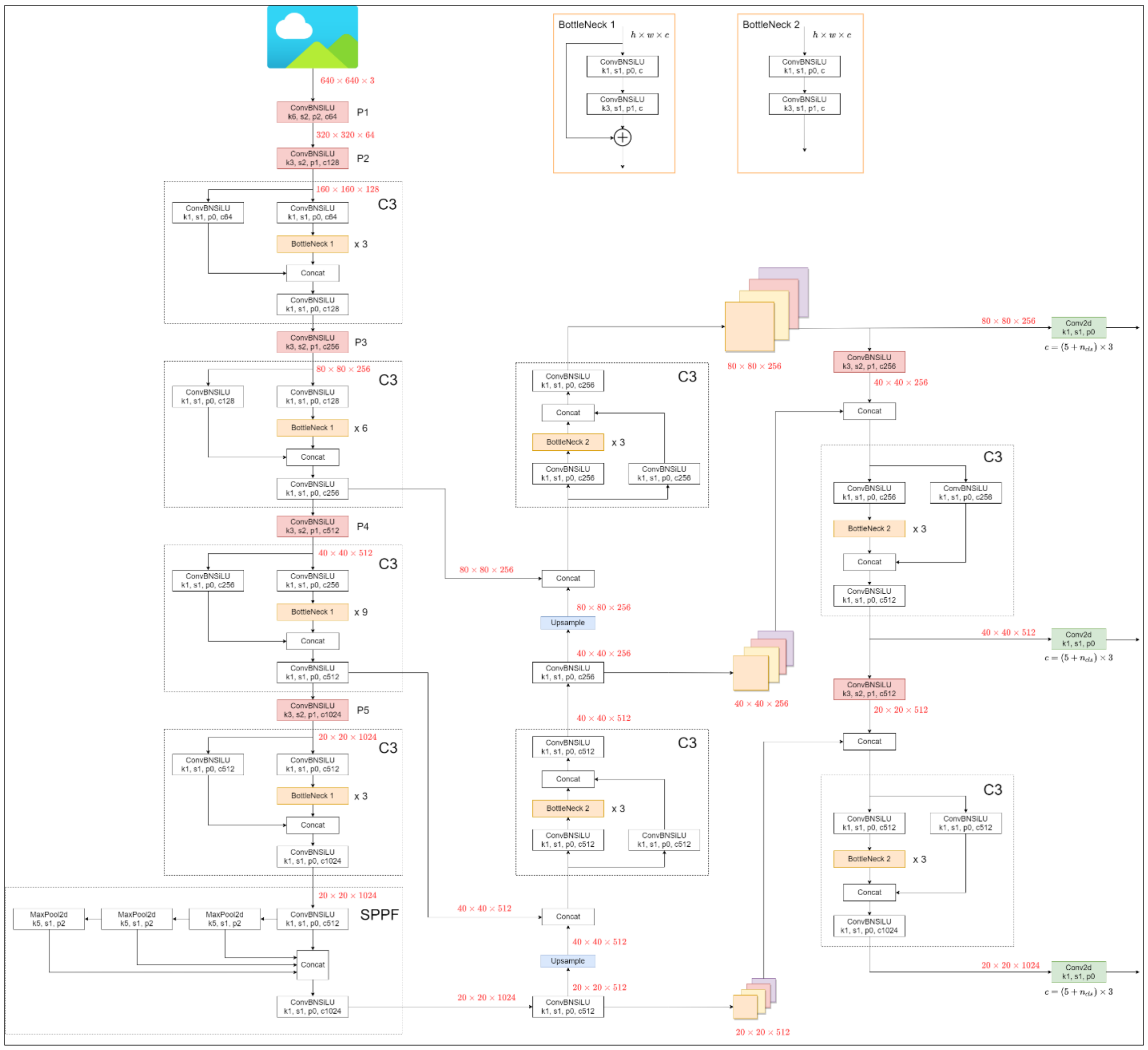


Figure A.1 YOLO V5s model architecture, Source: GitHub (Jocher, 2020)

APPENDIX B

TRAINING RESULTS – OBJECT DETECTION ALGORITHM

Additional results on the performance of the transfer learning algorithm for object detection are included in this section.

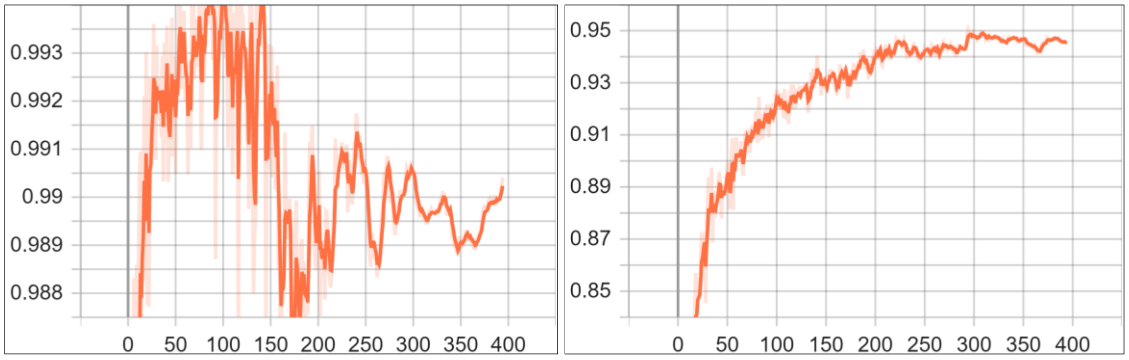


Figure B.1 Mean Average Precision (mAP) vs. Number of Epochs for (Left) Intersection over Union (IOU) = 0.5, and (Right) IOU = 0.5-0.95

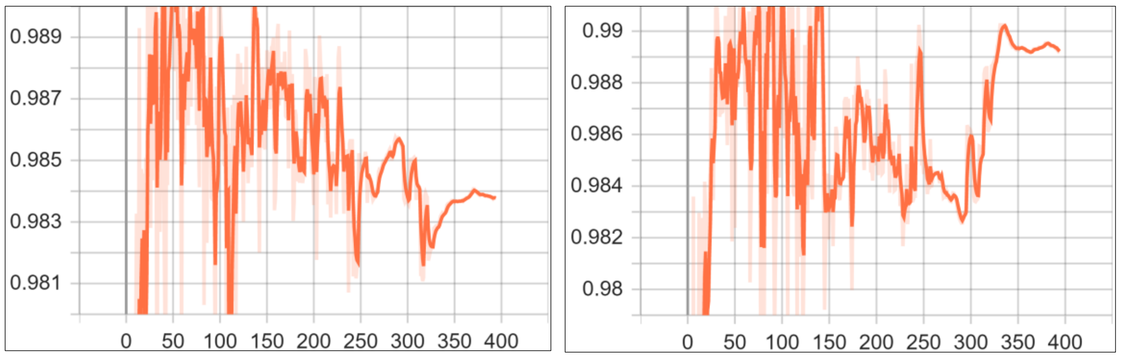


Figure B.2 Metrics Precision vs. Number of Epochs (Left) and Metrics Recall vs. Number of Epochs (Right)

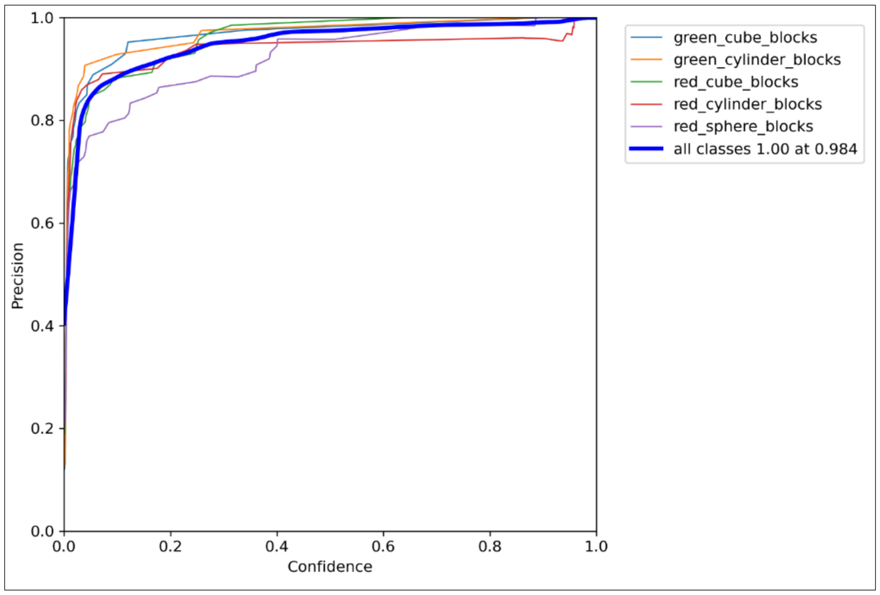


Figure B.3 Precision Curve

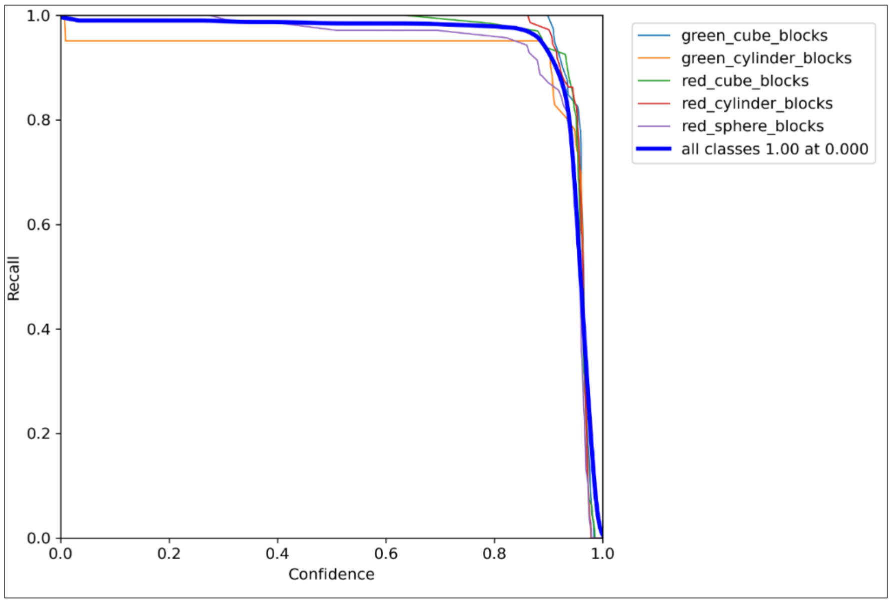


Figure B.4 Recall Curve

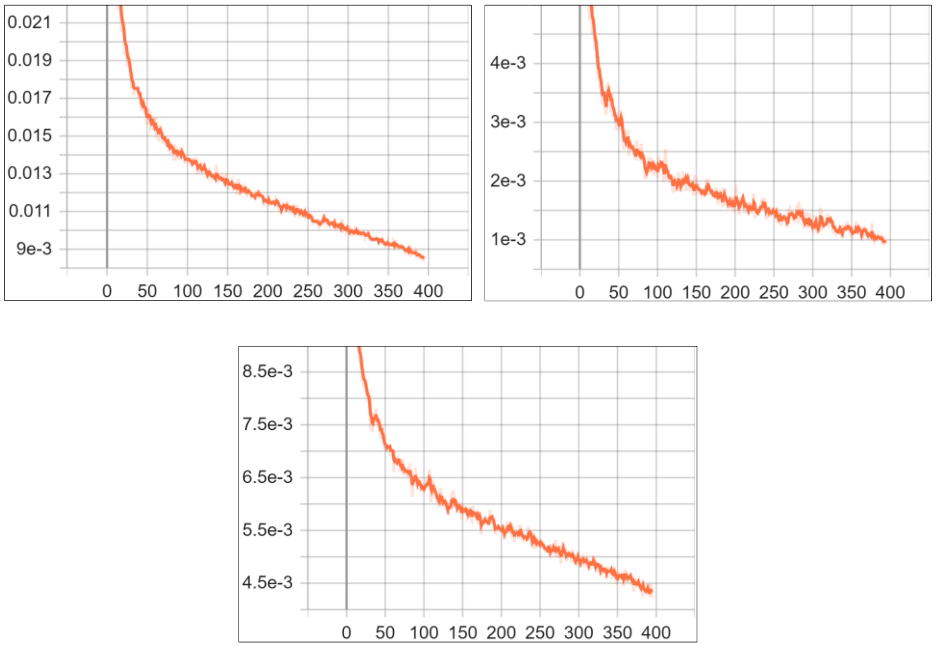


Figure B.5 Training: Box Loss (Top Left), (b) Class Loss (Top Right), and Object Loss (Bottom) vs. Number of Epochs

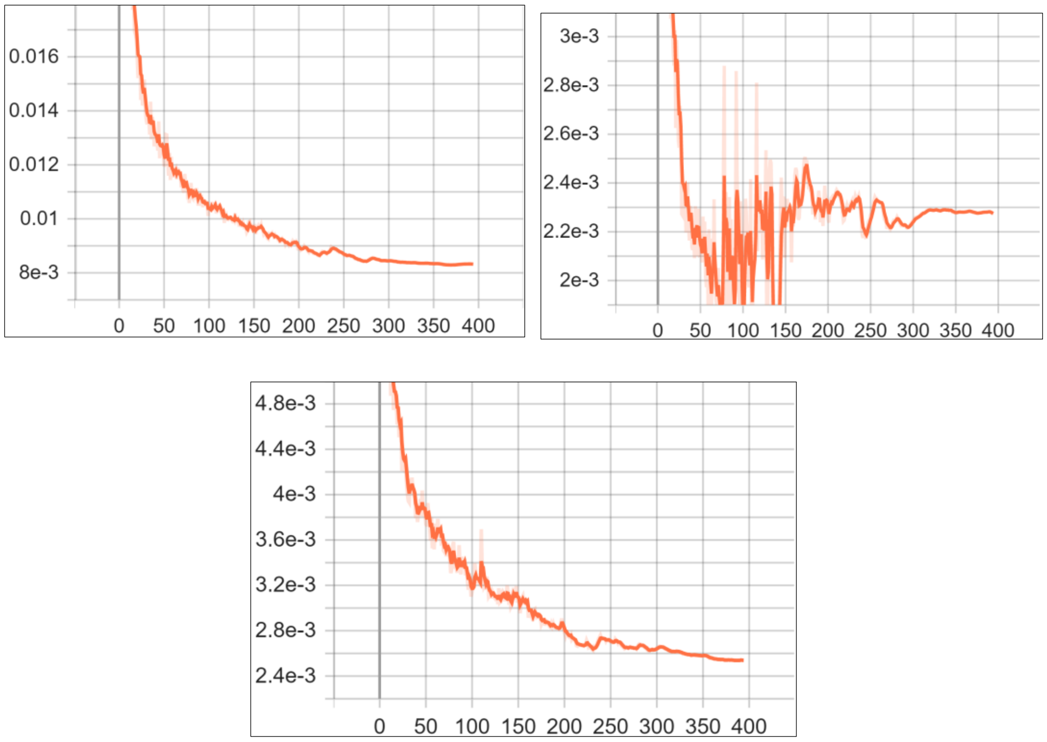


Figure B.6 Validation: Box Loss (Top Left), Class Loss (Top Right), and Object Loss (Bottom) vs. Number of Epochs