Sensor Directed AI-Based Robot Task Planning
in Unstructured Environments

By

ALIREZA MOUNESISOHI
DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Mechanical and Aerospace Engineering

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Prof. Bahram Ravani, Chair

Prof. Paul Erickson

Prof. Zhaodan Kong

Committee in Charge

Year of Degree (2022)

Abstract

This dissertation aims to improve methods for sensor-guided intelligent robotic task planning in unstructured environments. It will examine two fully unstructured environments: roadway maintenance and maintenance of space habitats. Maintenance automation of roadways and space habitats using robotic systems requires sensors and intelligent planning since both environments are highly unstructured. Robotic systems for automating roadway maintenance functions are typically installed on moving maintenance vehicles and must account for the unstructured nature of roadway traffic and conditions. In space habitats, when astronauts are not present, disturbances can perturb the environment, requiring sensor-driven and task planning robotic systems to deal with the habitat changes when performing maintenance functions.

This dissertation first considers the automation of roadway crack cleaning or sealing using a vehicle-based robotic system. It develops a machine-vision based task planning system to automatically detect and map roadway cracks and guide a manipulator following a crack on the pavement. The work in this area is limited to roadway edge cracks that are approximately straight compared to alligator pavement cracks. The results are demonstrated experimentally on a linear device used to mimic the motion of vehicle-based robotic systems.

Space habitats are very structured until a perturbation occurs due to a malfunction or movement of items within the habitat or in robotic interactions with on-board manufacturing as new parts are being fabricated. Robotic interaction with 3D printing manufacturing in space habitats is the second unstructured environment considered in this dissertation. A sensor-directed task planning and simulation method is developed which can autonomously fine-tune a robotic motion plan during interactions with the objects to be handled and address changes in objects configurations due to perturbations that can occur in the robot task space. As a result, this sensor-directed robotic system with task planning capabilities can be used for simulation and verification of several On-board Manufacturing (OBM) and maintenance operations.

Acknowledgements

First and foremost, I would like to thank my advisor Bahram Ravani for suggesting the topic of this dissertation as well as for his consistent guidance, encouragement, inspiration, and leadership during my Ph.D. program. Bahram's immense experience and determination in many branches of science motivated me to explore new areas that interest me. Without his insight, it would have been nearly impossible to overcome challenges of getting this doctorate degree.

I am grateful for all the generous funding support from the University of California, Davis, including the awards, grants, and scholarships. I want to thank professors: Paul Erickson, Donald L. Margolis, Zhaodan Kong, Ronald A. Hess, and Yong Jae Lee whose classes I took or from whom I received critical feedback to complete my dissertation.

I want to thank the Advanced Highway Maintenance and Construction Technology (AHMCT) Research Center. I have obtained ongoing financial support and equipment from the center. Working at AHMCT has also provided the opportunity of getting to know amazing colleagues. In that regard, I would like to thank Travis Swanston, Duane Bennet, Dr. Ty Lasky, and Dr. Sean Donohoe.

I would also like to thank the organizations that provided additional support during my program: the AHMCT research center funded by the California Transportation (Caltrans) providing funding for the first portion of this work and Habitats Optimized for Exploration Missions (HOME) program funded by the National Aeronautics and Space Administration (NASA) providing the second portion of this research.

I would like to express my gratitude to my family from whom I've received a great deal of emotional support. Finally, I would acknowledge my cousin and all the medical staff who worked hard during the pandemic and saved many lives.

"Asimov's Three Laws of Robotics:

1. *A robot may not injure a human being or, through inaction, allow a human being to come to harm.*
2. *A robot must obey orders given it by human beings except where such orders would conflict with the First Law.*
3. *A robot must protect its own existence as long as such protection does not conflict with the First or Second Law."*

Isaac Asimov ~ 1920-1992

Table of Contents

List of Figures

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

List of Tables

List of Acronyms

2D – 2-Dimensional

3D – 3-Dimensional

AC – Asphalt Concrete

AHMCT – Advanced Highway Maintenance and Construction Technology

AI – Artificial Intelligence

API – Application Programming Interface

AT – Adaptive Threshold

BGR – Blue-Green-Red

CAD – Computer Aided Design

Caltrans – California Department of Transportation

CCD – Charge-Coupled Device

CNN – Convolutional Neural Network

DCNN – Deep Convolutional Neural Network

DDS – Data Distribution Services

DOF – Degree of Freedom

DS1 – Deep Space 1

DT – Digital Twin

EVA – Extravehicular Activity

EVR – Extravehicular Robotics

FN – False Negative

FP – False Positive

FPN – Feature Pyramid Network

GigE – Gigabit Ethernet

GUI – Graphical User Interface

HDMI – High-Definition Multimedia Interface

HOME – Habitats Optimized for Exploration Missions

HSV – Hue-Saturation-Value

ID – Identification

IoU – Intercept over Union

ISAFR – In-Space Assembly, Fabrication, and Repair

ISS – International Space Station

IVA – Intravehicular Activity

LEO – Low Earth Orbit

LiDAR – Light Detection and Ranging

LNS – Least Normal Square

mAP – mean Average Perception

NASA – National Aeronautics and Space Administration

NCHW – N-Channels-Height-Width (A Data Format)

NMS – Non-Max Suppression

NRC – National Research Council Canada

OBM – On-board Manufacturing

OC – Line of Organic Correlation

OLP – Off-line Programming

OLS – Ordinary Least Squares

PCC – Portland Cement Concrete

PID – Proportional-Integral-Derivative

POSIX – Portable Operating System Interface

RA –Remote Agent

RGB – Red-Green-Blue

RGBD – Red-Green-Blue-Depth

RMSE – Root Mean Square Error

ROI – Return of Investment

ROS – Robot Operating System

SPDM – Special Purpose Dexterous Manipulator

SRDF – Semantic Robot Description Format

SRMS – Shuttle Remote Manipulator System

SSD – Single Shot Detector

SSRMS – Space Station Remote Manipulator System

ST – Simple Threshold

TCP – Transmission Control Protocol

TN – True Negative

TP – True Positive

TPU – Tensor Processing Unit

TTLS – Transfer Tank Longitudinal Crack Sealer

UR – Universal Robot

URDF – Universal Robot Description Format

USB – Universal Serial Bus

YOLO – You Only Look Once (Real-time Object Detection Architecture)

CHAPTER 1

1. Introduction

CHAPTER 1

1.1. Background

The research on developing sensor-directed robotic manipulators working as autonomous intelligent systems in real world environments involves a necessary combination of practical and theoretical considerations. The practical aspect plays a significant role in information processing to determine the most cost-effective and efficient use of resources. Autonomous intelligent systems are the integration of several subsystems working together to attain certain goals efficiently and safely. Integrating task planning for sensor-directed robotic systems is also important from a theoretical perspective. The computation components of intelligence are one of the critical targets of artificial intelligence (AI) research [1]. Advanced AI-based approaches can be included in perception systems to improve accuracy and self-sufficiency.

Most if not all autonomous systems using automated planning are built individually based on subsystems [2]. As shown in Figure 1-1, the major subsystems are sensing modules, actuation modules, and computation modules. Examples of the sensory modules include cameras, Light Detection and Ranging (LiDAR), force sensors, proximity sensors, and radars. The use of actuators for robotic arms and manipulators is essential for the movement of a robotic system as well as for changing the state of the environment. Finally, the intelligent system needs a brain to compute and analyze data from the perceptual system as well as plan and execute commands to achieve the goals. This computational aspect includes system awareness, signal processing, control, and planning. Robotic applications require sensors, actuators, and computational systems to be well-integrated to conduct high-precision operations. Well-defined intelligent automated systems are beneficial for factories, transportation industries, and space exploration missions [3].

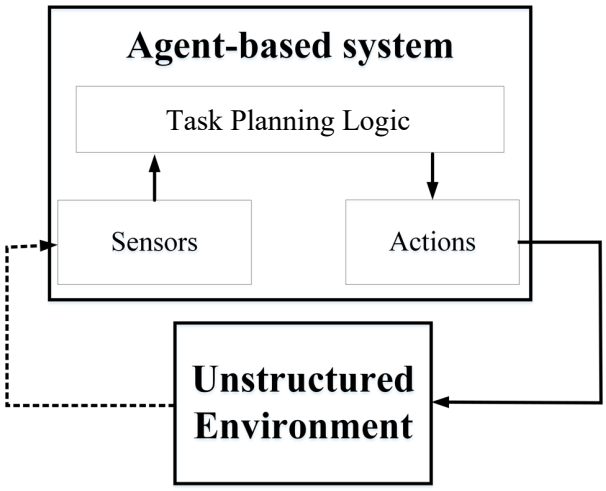


Figure 1-1 Basic diagram of sensor-directed agent-based robotic systems.

The environment in the real world is generally unstructured, meaning that unexpected circumstances may occur. Therefore, autonomous robotic systems require integration of sensors with machine learning and task planning capabilities to cope with dynamic environments. This dissertation aims to improve methods for sensor-guided intelligent robotic task planning in unstructured environments. It incorporates advanced model-based task planning techniques into autonomous robotic maintenance systems to increase their self-sufficiency. It will examine two fully unstructured environments: roadway maintenance and space habitat maintenance. Automating maintenance of roadways and space habitats using robotic systems require sensors and intelligent planning systems since both of these environments are or can become unstructured. Robotic systems for automated roadway maintenance functions are typically installed on moving maintenance vehicles and deal with the unstructured nature of roadway traffic and conditions. In space habitats, when astronauts are not present, disturbances can perturb the structure of the environment, requiring sensor-driven task planning robotic systems to deal with the changes when performing maintenance functions.

This dissertation first considers the automation of roadway crack cleaning or sealing using a vehicle-based robotic system. It develops a machine vision-based task planning system to automatically detect and map

roadway cracks and guide a manipulator to follow a crack on the pavement. The work in this area is limited to roadway edge cracks that are approximately straight compared to alligator pavement cracks. Longitudinal highway edge cracking is considered one the most destructive forms of pavement defects as it absorbs 75 percent of the total water entering highway sublayers [4]. Because highway edge cracks are relatively straight, automated robotic operations on them can be continuous, i.e., the robotic system can run without coming to a complete stop. In this dissertation the application of the vision-based task planning system developed is demonstrated in an experimental testbed that can mimic an actual robotic system working on a roadway environment.

Second, this dissertation develops a model-based task planning system for sensor-guided robotic applications in space habitats with interactions with OBM. The structured nature of a space habitat allows for developing a baseline digital world model that can be used in a model-based task planning system. This dissertation develops a digital model of an example space habitats that can provide the basis for a robotic task planning simulation system. OBM can fabricate new parts and present them for robotic handling that would require on-the-fly sensing their locations and features. This dissertation develops a sensor-directed task planning system to handle robot interactions with an on-board 3D (3-Dimentional) printer. The US lab of the International Space Station (ISS) is used as an example. The combination of the simulation environment and the sensor-based task planning system can be used human integrated robotic task verification as well as for autonomous task planning. The simulation of the use cases performed in this work shows that the robotic task planning system developed is able to increase its autonomy significantly, gaining the ability to query the state of the environment by observing objects, to communicate with other objects, to conduct automated path planning, and to handle newly printed parts from 3D printers.

The contributions in this dissertation include enhancing robotic task planning and model-based programming for sensory robotic systems. The first contribution is the use of machine vision technology for a vehicle-based robotic guidance system. The automated system employs a state-of-the-art deep

convolutional neural network (DCNN) to detect object features and extract their actual coordinates. These image features are translated to the world frame and then used to compute the error for feedback control of the robotic end-effector. The application of DCNN-based image processing for a roadway crack detection and cleaning or sealing operation is new. The method, when integrated with a vehicle-based robotic actuation system, allows for self-guided operations for manual or automated edge crack sealing or cleaning operations on a continuous basis. The method developed is illustrated in a simplified experimental system.

In the second part, this dissertation develops a simulation system for robotic task planning, programming, and verification purposes. This simulation system uses Gazebo physics engine, the Moveit Application programming interface (API) and the Robot Operating System (ROS)-based, off-line program. In addition, the task planning system uses the planning technique for high-level control of abstract commands to interact with objects in a 3D printer.

1.2. Dissertation Structure

The structure of the remainder of the dissertation is as follows:

Chapter 2 reviews the current state of sensor-based robotic task planning systems. Furthermore, it evaluates the literature on sensor-guided robotic guidance system applications. Roads and space stations are examined in two different instances. Finally, this chapter provides an outline of some of the relevant research gaps that will be addressed in the subsequent chapters.

Chapter 3 presents the development and implementation of the sensor-directed guidance system with advanced machine learning techniques in roadway crack cleaning or crack sealing applications. It presents the actuation's control logic and task planning to track longitudinal cracks through a linear manipulator. It then covers the evaluation of the system inside and outside of the laboratory setting.

Chapter 4 presents the development of the simulation system and the robotic task planning and automation of in-space manufacturing using a sensor-guided intelligent robotic systems. The challenges of task planning sensor-directed robotic guiding and interaction with 3D printing in space is analyzed and addressed.

Chapter 5 provides some of the conclusions of this research focusing on the functional connections between the systems developed for the two different applications one for roadway maintenance and second for space habitats as well as discussing some of the areas of future research.

CHAPTER 2

2. Overview of AI Task Planning

CHAPTER 2

2.1. Introduction

An intelligent robotic system is one that can perform tasks autonomously or independent of human input in a given context [2]. Robots require task planning algorithms to automate the execution of actions necessary to achieve goals that are impossible to accomplish through individual actions. The system designer must consider two primary aspects in developing a robotic task planning system for the system to be used effectively. The first aspect is to develop individual subsystem components [2]; the second aspect is to integrate those components into a unified model that will work together to achieve a goal [2]. These two processes are interconnected, and system engineering expertise is required to create a fully functional robotic system with efficient communication between the system components.

Components which commonly make up task planning systems are sensor modules, actuation modules, a logic unit, and a command interface. In this architecture, the logic unit is the primary component and accepts commands from, and communicates status to, the command interface. To perform task planning and execution, the logic unit analyzes data from the sensor modules, computes a plan, and controls the robots via the actuation modules. Proper integration of these components would allow the autonomous completion of tasks by the robot agent, potentially with a high level of autonomy.

2.2. Task Planning

An automated task planning system must be capable of developing task plans using its computing module. Data driven task planning systems may be broadly classified into two categories: pre-programmed for a given task objective and autonomous with world sensing and task analysis in which the robot plans and executes a set of actions based on sensory input. The primary distinction is whether a robot's actions are determined by a detailed sequence of steps provided to it before performing a task or whether a robot

functions autonomously by intelligently acquiring and interpreting sensory data, perceiving the world, changing its work environment, and planning its actions. In most cases, providing a robot with a set of actions to deliver a task in an environment that constantly changes is impractical. Frequently in these cases, the system encounters new and unexpected events and fails the operation. Therefore, these systems have to demonstrate significant levels of autonomy to cope with new, upcoming uncertainties. These uncertainties may be accomplished with robot's task planning and advanced communication protocols between the robot's agents, such as sensors capable of mapping the world and passing information on to plan, control, and execute tasks[5]. Constant dependability under uncertain real-world situations is a significant issue in developing intelligent robotic systems [5], [6].

2.2.1. Task planning in unstructured environments

An intelligent robot system with autonomous capability is not guaranteed to function in every environment[7], [8]. Since the environment of the robot is application-based, it is best to concentrate system perception and data processing in particular settings. The challenges from perception to actuation and all the needed task planning for every application is inherited in the nature of the task to which a system is designed to operate in [7]. For instance, the development of the architecture of a sensor-directed task planning for an intelligent aerial device is completely different from that of an autonomous vehicle. The automated task planning system must extract related features from the observations of the perceptual system and use them to accomplish the goal's task to compensate for the uncertainties of the environment [7]. Some examples of these considerations include the need to model the world or not, if only one robotic system is operating or multiple robots are operating, how often changes occur in the environment, and how the controller should be engaged in the system.

2.2.2. Task planning and offline programming

The development of discrete components that can be used in a larger robotic system has been the focus of research investigations over the past decade [9]. These research studies contributed to the development of efficient image processing, analysis, and interpretation techniques as well as several robot control and path-planning systems, digital representation modeling, task planning, and off-line programming[10]. Off-line programming (OLP) is a robot programming technique in which the program is written apart from the physical robot in a virtual setting [11], typically using a CAD based simulation environment. The robot software is subsequently uploaded and executed on a physical industrial robot [10].

Autonomous robotic operations involve robotic actions without human input. This can be achieved, for example, in a sensory environment when the computational module of a robotic system can intelligently process sensory data and modify a pre-planned task and develop new actions to complete a task effectively (see [11], [12]). In such a system, the sensors identify local abnormalities from the ideal state, and the robot system handles them autonomously[10]. As a result, sensor-controlled motions must be employed to elevate local autonomy to the level of the machine[9]. For complete autonomy, high-level planning capabilities for task planning and intelligent error management are required, but current methodologies are insufficient to provide the necessary tools[10]. At the moment, complete autonomy does not seem to have been achieved in many robotic applications.

2.2.3. AI task planning in nature

The natural task planning behavior of an Alaska brown bear while fishing in a river can illustrate the purpose of planning in unstructured environments. The bear's fishing process involves three steps: orientation, approach, and capture [13]. The bear takes several positions which are sitting, standing, and walking. There are also several approaches that the bear can take to reach the fish, such as loping, head underwater, and plunging. Finally, to capture the fish, it can use its forepaws, mouth, or both. To eat the fish, the bear may

take any possible options in orientation, approach, and capture. For example, it may use standing- plunging-forepaws or sitting-head underwater-mouth. Although the underlying reasoning behind the selection of a bear's fishing strategy is unclear, it is evident that bears take combinatorial tactics every time they near a river to hunt. For example, a hungry bear, approaches the environment, selects a strategy, waits, searches, and tracks for the prey, and uses its mouth or forepaws to seize the fish.

a)

b)

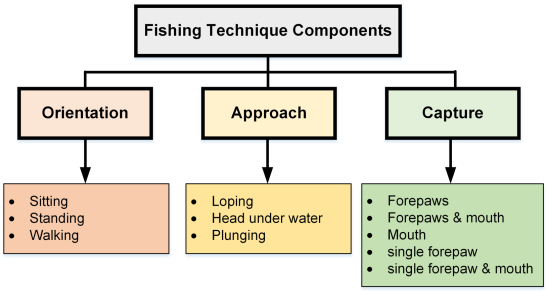


Figure 2-1 a) Bear fishing[14] exhibiting single forepaw and mouth to capture b) Bear's fishing technique components[13]

Alaskan bears' fishing behavior, however, is alike intelligent robotic systems. First, the bear is hungry and needs food. This desire is assumed to be the highest-level command in this example[13]. When arriving at the river, the bear knows that the prey is close. The bear then searches and tracks the fish in the creek, which is similar in concept to intelligent robotic systems utilizing the sensory motors of its perceptual system to search and track objects. The bear plunges its head in the water to increase the chance finding a fish, and the bear tracks the prey until it is within reach. Finally, using either its mouth or forepaws, it hunts the prey. In robotics, this approach of tracking is referred to as visual servoing, and hunting is similar to the concept of reactive action[15]. The ultimate goal of an intelligent system is to achieve its goals. The example of the Alaska bear provides a clear picture of how small actions are organized to achieve a goal, which can then be applied to many application-based robotic systems.

2.2.4. Robot task planning in maintenance

Unstructured environments are referred to the conditions that are subject to change[15]. Such environments can be common when performing maintenance in a system. Maintaining a system or an environment occurs to restore the system to a desirable condition[16]. Currently, the number of complex environmental systems in need of monitoring and maintenance is increasing[17]. The maintenance function can be divided into three types of actions: inspection, regular maintenance, and disturbance handling [18]. The first type of action, inspection, is to assess the environment. The second maintenance action, routine maintenance, is about scheduling a system to be checked to prevent considerable damages to the system. The third maintenance action is the disturbance handling for sudden failures of the system. The design of an intelligent system should consider all the maintenance actions. Designing intelligent robotic systems for any of these maintenance actions is a subject of interest for research agencies and companies[18]. The automation of these maintenance processes would have a long time Return of Investment (ROI) and would even allow some currently not yet conceived missions, such as space exploration, to be carried out[18]. In the following sections, an overview of the challenges of system design and planning for two different environments is discussed. The first environment involves roadway maintenance dealing specifically with robotic crack filling/sealing operations. The second environment involves robotic applications in space habitats. The remainder of this chapter provides an overview of the some of the related works in robotic task planning.

2.3. Related Works in Task Planning

2.3.1. Traditional task planning

Some of the existing architectures for robot control systems separate sensing and perception sub-system from the planning and action sub-systems. The two sub-systems then interact via an explicit world model

[19]. Figure 2-2 provides a schematic of such an architecture. The sensing and perception sub-system monitors the environment and maintains a comprehensive representation of the status of objects in its vicinity. The planning and acting sub-system derives its perception of the world from the model and chooses actions in response to the perception system.

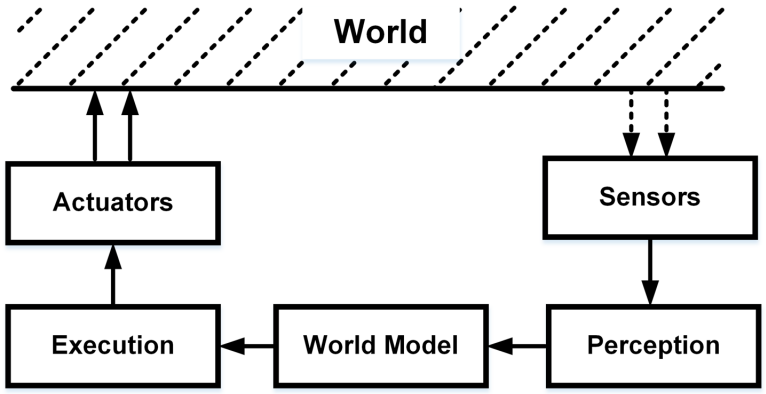


Figure 2-2 Traditional task planning architecture model inspired from [19]

The separation of planning and perception allows for handling the two aspects of planning independently [19]. Previously, planning research focused on the challenge of developing a series of basic robot operations to fulfill a set of objectives using a comprehensive initial world model as a point of reference [19]. Currently, due to the asynchronous nature of events such as arrival of other robots and the presence of invisible barriers, the model must include sensor data throughout task execution [8].

A challenge that emerges when planning and perception are separated is that the perception sub-system must create the world model without knowing the robot's goals [15]. An environment may encompass hundreds of different items, and the sensing system must be readily available to provide information about the status of each one for the planner [19]. If the planning system communicates with the sensing system exclusively via the world model, the sensing system has no means of knowing which features the planner

needs[15]. Thus, sensing research has been focused on continuously describing the exact condition of everything in the world [20].

2.3.2. Task Directed Sensing

The concept of task-directed sensing has been studied for several decades. A separate world model, in particular, is an essential element of the design as displayed in Figure 2-3. As can be seen in this figure, the world model contains the most up-to-date information that the perception system has about the actual environment, including the current values of all sensors as well as prior recordings of what those sensors detected [8]. In this architecture, the planning system directs the sensors and the analysis of their data [19]. The planner is tasked with analyzing the robot's objectives and the present state of the world model and producing both the action and sensing tasks necessary to accomplish the objectives.

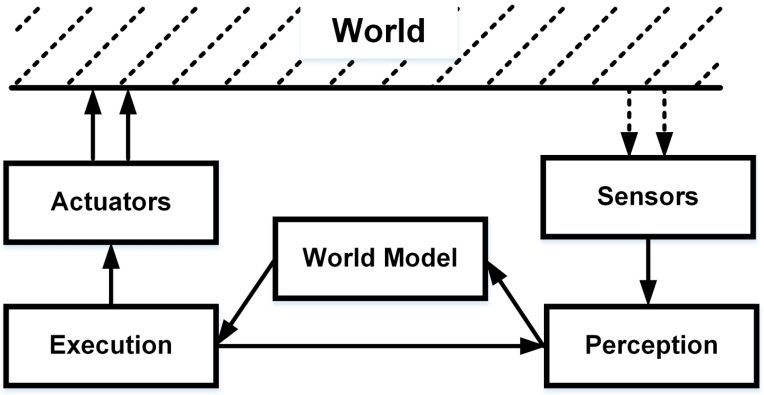


Figure 2-3 Task-directed sensing planning model inspired from [19]

Additionally, task-directed sensing implies that the most effective method for incorporating sensing functions into plans is to consider them as action tasks. As goals are decomposed into executable actions, the sensing functions necessary to support those actions are added. Sensing tasks, like actions, can be incorporated into planning hierarchies and templates or they can be represented as separate operators that change the state of the world model [19]. In other words, actuator tasks work in the world whereas sensor

activities operate on the world model. In this manner the future system of the world model can be anticipated in the same fashion as the future state of the world can be expected. Actions modify the world, and sensing tasks modify the robot's understanding of the environment [21].

2.3.3. Reactive Execution Method

One technique for joining robotic perception and planning is to combine them into a unified, state-free, entirely reactive system [19]. This is depicted in Figure 2-4 where the sensors provide information for the decision network which guides the actions of the actuators. The motivation for this method is to address a significant issue that occurs when an agent's world model is inadequate or uncertain. In such a situation plans built on that world model will be unreliable [22]. Due to the perception system's inability to maintain an ideal world model, plans are inevitably inaccurate a significant portion of the time [23]. Plan execution eventually fails, and new plans are needed to generate pre-existing world model failure [23].

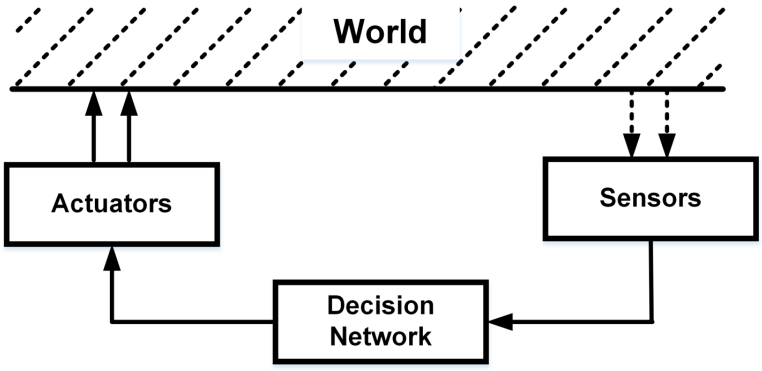


Figure 2-4 Complete reactive task planning model inspired from [19]

Reactive execution systems solve this issue by producing just one action at a time and selecting it based on the world's current state, not on a model of that state [19]. The goal is to reduce the inherent uncertainty in a model by simply relying on directly observed input. Reactive type systems directly link a robot's sensors

to its actuators via a decision network that enables the current action depending on the sensors' current readings, or sensor data indexes straight into an action to perform [19].

2.3.4. Deliberation in Task planning

During the deliberation decision process, the robot selects and organizes activities by predicting intended consequences [1]. This deliberation is used to achieve as many predetermined objectives as possible [24]. The state of the dynamic system is constantly changing. Two factors cause these changes: events and actions. Events are environmental occurrences that supervisory planners cannot control or modify [1], [24]. On the other hand, actions are the environmental occurrences that the principal planner can execute at the right time [25]. In this environment, the agent is an entity that can perform actions; however, the agent does not necessarily reason about its actions. If it does reason about its actions and anticipates the outcomes, it is called an actor. The performed action by agents can take several forms, including an exerting force, a movement, activation of sensory motors, and communication [1].

2.3.5. Automated Task Planning

Automated task planning computationally examines the deliberation process of robotic systems [21]. The interplay of the three components consisting of observations, plans and actions is shown in Figure 2-5. Given a world system's state denoted by "s" as input, a controller outputs an action denoted by "a" following some plans [21]. A state-transition function denoted by "ϒ" is used to transform these components in response to the events and actions that it receives [1]. Describing the system, a starting state, and an aim, a planner generates a plan for the controller to follow to accomplish the objective. There are also events denoted by "e" that occur in the world that can impact the plans. Each of these four components of planning namely s, a, e, and ϒ have a set of values where their sets are denoted by S, A, E and ϒ. A state transition system model can therefore be represented by Σ [21] consisting of these four sets as follows:

$$\Sigma = (S, A, E, \Upsilon)$$

Where:

- $S = \{s_1, s_2, s_3, \dots\}$ is a limited or iteratively enumerable set of states;
- $A = \{a_1, a_2, a_3, \dots\}$ is a limited or iteratively enumerable set of actions;
- $E = \{e_1, e_2, e_3, \dots\}$ is a limited or iteratively enumerable set of events; and
- $\Upsilon = S \times A \times E \rightarrow 2^s$ is a state-transition function.

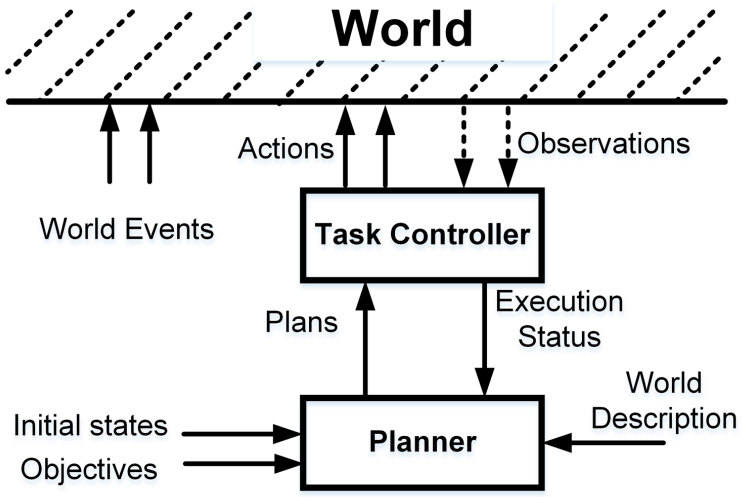


Figure 2-5 A conceptual model for dynamic planning

2.4. Sensor-directed task planning method

The primary prerequisite for autonomy is the capability of intelligent sensor data processing to guide completion of a task effectively. Actual sensory data are typically compared to a specified reference pattern to provide information to the robot controller to attain the desired end position [19]. The sensors identify local abnormalities from the ideal state, and the robot system handles them autonomously [7]. As a result, sensor-controlled motions must be employed to elevate local autonomy to the level of the machine [26].

For complete autonomy, high-level planning capabilities for task planning and intelligent error management are required, but current methodologies are insufficient to provide the necessary tools [24].

2.4.1. Sensor-directed AI-based robot task planning

In this dissertation, the idea of sensor-directed task planning was developed in response to realistic sensor deployment requirements. It tackled the issue of timely delivery of critical information to the planning and action system. Because it is only feasible to have direct knowledge of a tiny portion of the environment at any given time, the sensing system must focus on the aspects of the world that are necessary for the tasks at hand. Sensor-directed task planning methods developed here use advanced system level communication methods, such as ROS [27], to connect the sensing and action systems. The planning and action system, in particular, guides the sensing system to gather and evaluate only information relevant to the robot's purpose.

Figure 2-6 shows the sensor-directed task planning architecture developed in this dissertation. In this architecture, the sensor and perception of the robot agent is separated from the world. This separation as shown clears the path for exact sensory information that can be used by the robot or to calibrate the model of the world. The perceptual system can be very sophisticated and incorporate several advanced data sources. The next element of this architecture is the task controller. The task controller is where the agent-based system receives the featured data from the perception system, aligns it with the current plan, and generates an action. The actions, once finally executed, will change the state of the world. The task controller is designed to behave reactively, which means that it helps to accomplish the task and compensates for local inconsistencies. Given the example of robot gripper grabbing an object, if the relative position of the object and the robot has slightly changed, the task controller takes care of the position change.

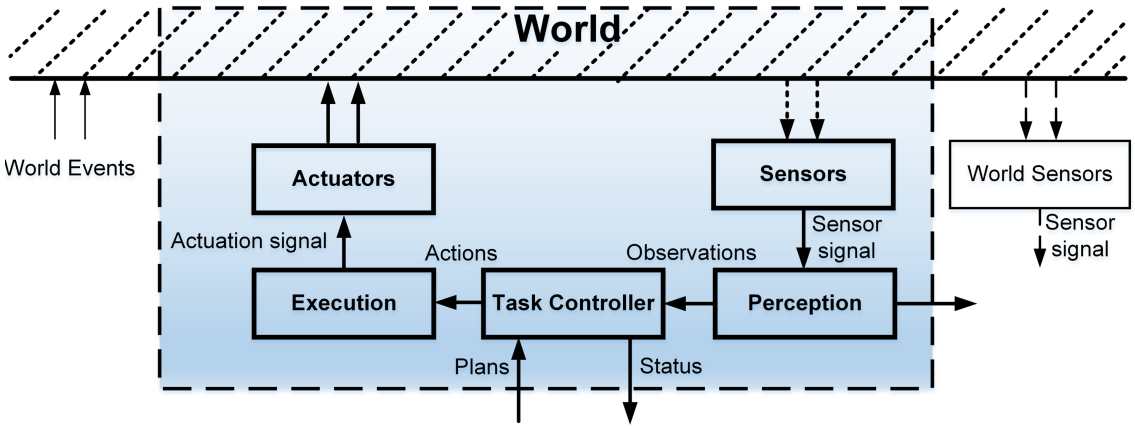


Figure 2-6 Sensor-directed task planning

Once the sensor-directed task planning is developed for every individual robot agent, then a central task planner can control a network of robots that interact with each other as conceptualized in Figure 2-7. Under the policy of the central task planning, each robot agent interacts with another agent and with the world. Each robot also receives its plan from the planner. In this scenario, every robot is helping to build the model of the world that is ultimately used by the planner to accomplish the goals of the operations.

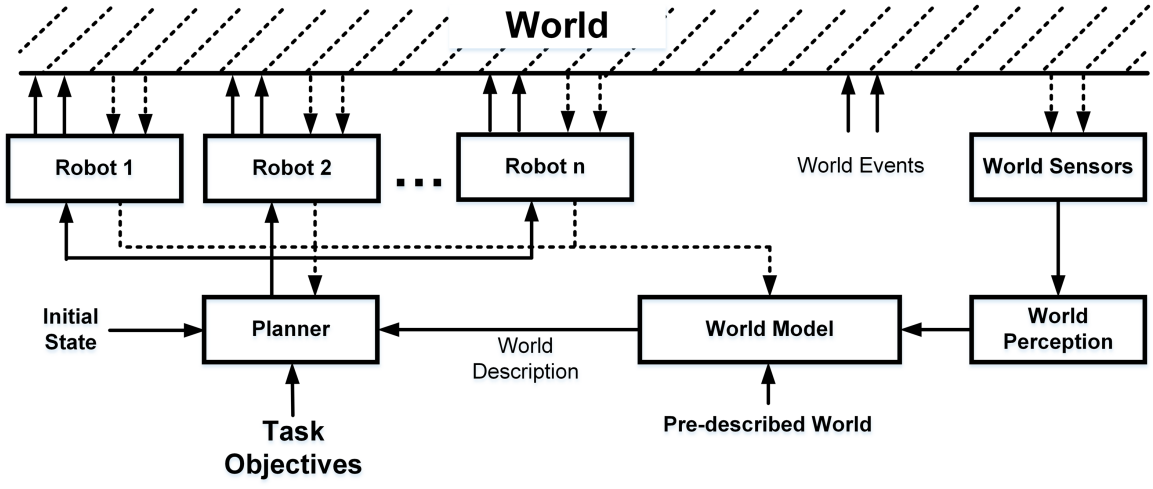


Figure 2-7 The conceptual sensor-directed task planning by separating robot agents from the world

In this dissertation, two scenarios are examined using this sensor-directed-task planning method. The first scenario is a vehicle-based robotic system to be deployed in an unstructured roadway environment. The second scenario is a robot arm-based system in the confined environment of a space habitat. Both of these systems use a robotic arm, or robotic guidance systems which will be discussed in chapters 4 and 5. The goal of the first guidance system in roadway environment is to track longitudinal cracks during operation. The goal of the second guidance system in a space habitat is to move objects, specifically 3D printed objects. Both of these environments are considered unstructured. In the roadway environment, vegetation and other debris may introduce noise to the perceptual system. In space objects, objects may move due to existence micro or no gravity or other events, leading to environment becoming perturbed or unstructured.

CHAPTER 3

3. Sensor-Directed Robotic Task Planning for Roadway Edge Crack Cleaning/Sealing Operation

CHAPTER 3

3.1. Introduction

Longitudinal highway edge cracking is one of the most damaging types of pavement defect since it absorbs 75% of all water entering the highway sublayers [4], [28]. Longitudinal/edge cracking runs parallel to the centerline direction of the road which are relatively straight as depicted in Figure 3-1. These edge cracks are very common in California highways consisting of 2,716 center lane miles of Portland Cement Concrete (PCC) highways. The total number of miles of edge cracks is twice to four times the number of center lane miles, since edge cracks can exist between the travel portion and the two shoulders, in both directions.

a)  b) 

Figure 3-1 Longitudinal joint crack on same highway in California a) In-lane longitudinal crack b) Shoulder Joint Transition crack

Robotic systems have been developed to automate the treatment of these longitudinal cracks (see, for example, [4], [28]). These systems consist of vehicles that travel the roadway with a connected robotic arm which applied sealant material to seal the edge cracks. Two such systems developed at Advanced Highway Maintenance and Construction Technology (AHMCT) research center at UC-Davis are the Transfer Tank Longitudinal Crack Sealer (TTLS) and Sealzall systems [4], [28]. Both of these systems use the driver to position a simple arm on the roadway crack and then apply the sealant material to seal the crack. Figure 3-2-a depicts the Sealzall system performing sealing operations of edge cracks on a California highway. A picture of the traffic control system used with the Sealzal system is shown in Figure 3-2-b. It consists of a shadow vehicle equipped with a crash attenuator following the Sealzal vehicle.

a) 

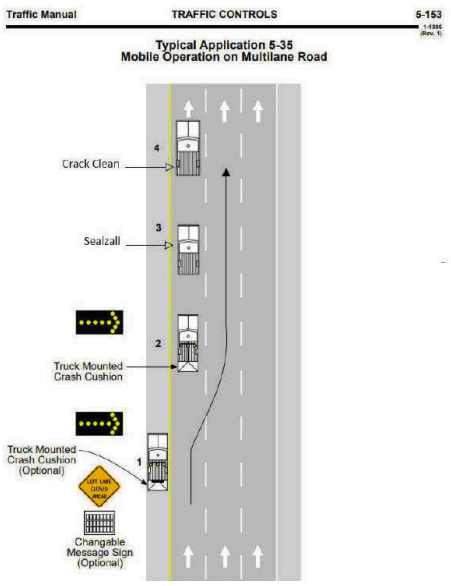
b) 

Figure 3-2 Vehicle-Based Sealzall Sealing a) Highway Traffic on Interstate b) traffic control flow

Both the TTLS and Sealzal system fully rely on the driver of the vehicle to properly position and guide the robotic arm over the cracks while driving the vehicle which would make the task difficult and exhaustive. As a result, there is a need for the development of a guidance system that can relive the driver for precise positioning and guiding the robotic applicator over the edge cracks while also driving the vehicle. In this dissertation, we develop a guidance system utilizing a vision based task planning approach to guide the robotic arm over the longitudinal cracks with both manual and automatic modes. This system decouples the driver from the robotic positioning function and allows the driver to focus on driving while the positioning function can be performed automatically by the vision based task planning guidance system or operated in human-in-the-loop mode by a separate operator.

The system developed here uses two cameras as perceptual sub-systems: a trailing-view camera and a heading-view camera. The task planning system is a supervisory system, which coordinates crack mapping, joystick inputs, and different operational modes. This task planning system uses Deep Convolutional Neural Network (DCNN) machine vision to detect and map highway cracks through the trailing camera and automatically guide the robotic arm over the pavement's longitudinal crack. After task planning

generates the action, it commands the robotic manipulator to properly position its end tip applicator. The end tip applicator is either a sealant device or a router so the system can be used for both crack sealing as well as crack routing/cleaning which is needed in some situations before the sealing operations. It should be pointed out that the guidance/task planning system developed is for edge cracks which are approximately straight and appear between the travelled roadway and shoulders. The system, in its present form, is not applicable to alligator cracks that appear across the travel portion of the roadway. The guidance system developed can be used with TTLS or the Sealzal systems or other similar systems and in combination provides for a roadway crack sealing maintenance system.

The roadway environment, when dealing with roadway edge cracks, is an unstructured environment due to the potential for uneven plant growth in the cracks as well as accumulation of unwanted debris. In addition, environmental conditions may impair crack sensing, such as rain, sun brightness, shade, or dim light. Since sensor data independently control actions by the robotic agent, the overall task planning system developed provides for different levels of autonomy based on its two modes of operations. A functional diagram of the corresponding maintenance system is depicted in Figure 3-3.

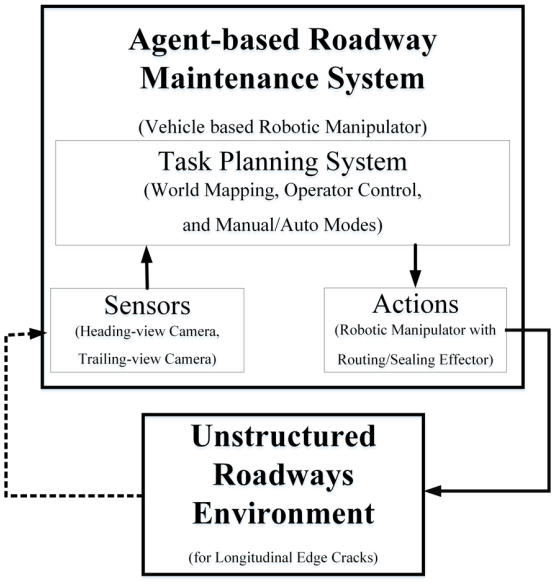


Figure 3-3 Functional diagram of the developed system for roadway edge crack maintenance

CHAPTER 3

Developing an automated guidance system capable of directing the cleaning/sealing end-effector of a robotic arm along the longitudinal joint crack creates two significant sensing challenges. The first challenge is due to the need for co-located crack sensing since the operational tool head is sufficiently big and can conceal the crack zone that must be sensed, obstructing the camera's vision. The second challenge is sensing under unstructured highway conditions.

To address the sensor co-location challenge, we installed the camera sensor in the closest trailing vicinity of the working zone in order to sense the extension of the longitudinal crack (see Figure 3-3). Figure 3-4 shows that the relative straight longitudinal gap divides the travel lane made of Portland Cement Concrete (PCC) from the shoulder made of Asphalt-Concrete (AC). The red-dotted square in this last figure is the region that is observed by the trailing camera. The black rectangle is where the router box conceals the crack section. Since longitudinal cracks are approximately straight, our crack sensing uses the non-concealed trailing image region behind the router box to detect and extrapolate the crack path. After the path is extrapolated through its position and orientation, then the lateral linear actuator moves toward the desired position where the cleaning or sealing operation should happen. Regarding the second problem, we selected a DCNN model and trained it over the real images of longitudinal edge cracks for robust detection approach to recognize longitudinal edge cracking on the highway shoulders.

a

b

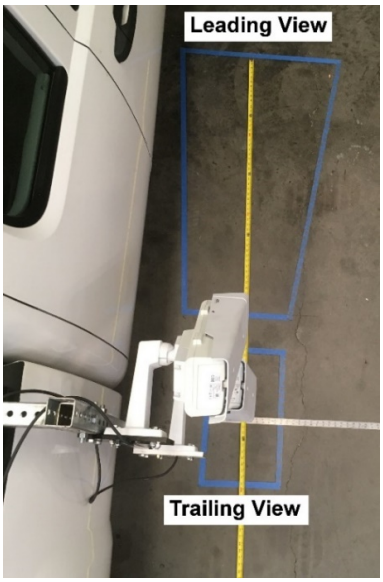


Figure 3-3 Installed positions of Leading and Trailing View cameras, a) Front-right perspective, and b) Top perspective

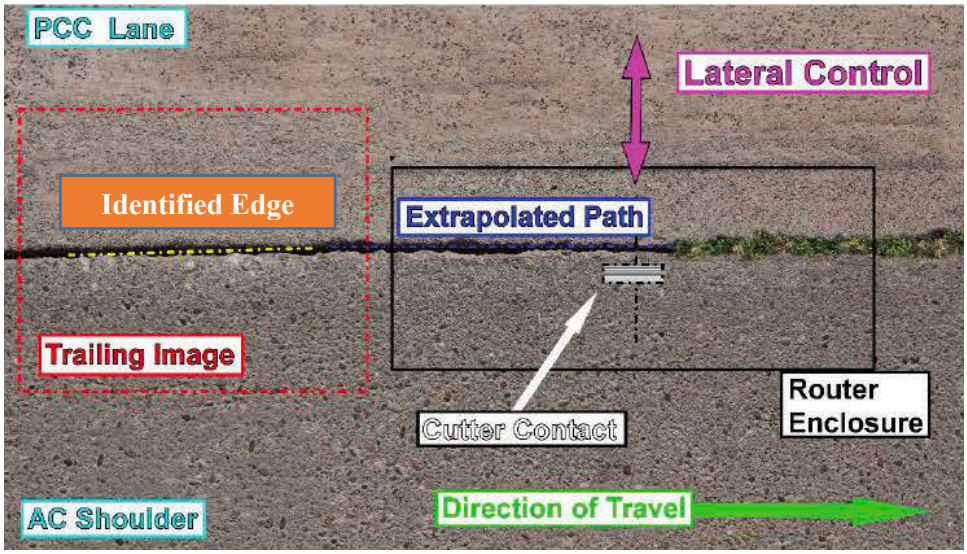


Figure 3-4 Automated co-located vision based longitudinal crack sensing

3.2. Automated Task Planning Supervisory System Control

The autonomy level of every system relies on hardware and software capabilities. From the design standpoint, some of these systems may have higher sensor and actuator capabilities. Also, one important factor is the cooperation with a human controller. In this section we develop a task planning framework

that can be controlled manually or semi-automatically. This system's task planning entails integrating several modules, such as joystick input, visual camera inputs, image processing, and operation modules. This integration must include a sophisticated logic algorithm capable of reconciling and combining these modules with their various data sources. The task plan controller, in principle, would handle the majority of the router steering action, which is the movement of the actuator. The supervisory task planner must watch for the coming joint cracks and involve the manipulator for positioning of the tool head. The planning program must also be in such that the operator is able to override the auto-mode control at any stage.

3.2.1. Task planning for a moving vehicle robotic system in a roadway crack sealing/cleaning application

The functional architecture of a sensor-directed task planning system for a robotic system on a moving vehicle performing work on a roadway is depicted in Figure 3-5. In such a system, the sensors provide information for the perception and task controller. They also send the data to the digital display for use by the operator and the driver. The operator can provide control inputs through the joystick to the execution module. The execution module then applies voltage signals to the actuator to perform the operations. The driver provides steering input to the vehicle. In crack sealing or cleaning operations, the task controller is obtaining data from the pavement longitudinal crack and sends the information to the perception system. The perception system is a multistage advanced deep learning system with feature extraction. The output of the feature extraction system is passed into three modules where, depending on the mode of the operation (manual control or semi-autonomous control, and automated control), only one of the three modules is activated.

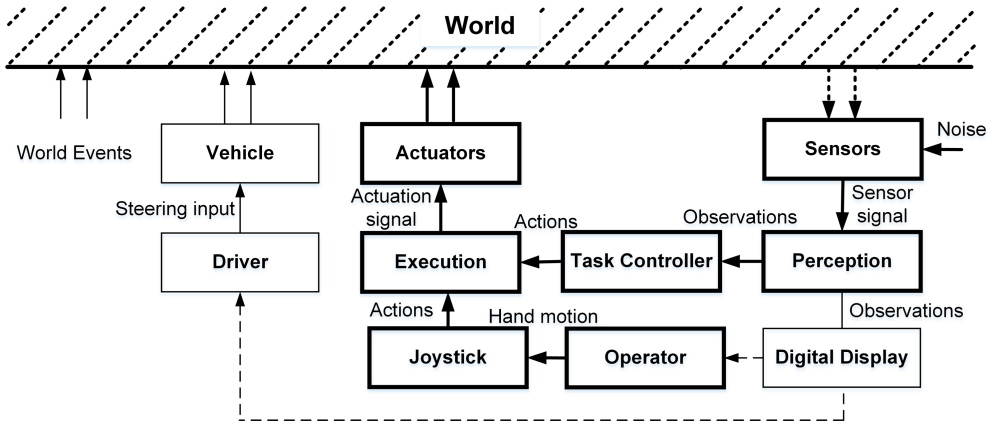


Figure 3-5 Sensor-directed task planning for a moving vehicle robotic in an unstructured environmental setting

The manual mode allows the operator to provide the needed input to the joystick to position the robotic end-effector on the crack. The operator can use the camera installed on the side of the vehicle to position the end-effector on the crack. The display system from image processing system can also be used which would provide a more enhanced image of the crack in a semi-automated mode. In the fully automated mode, the image data is passed through the deep learning-based image processing filters to provide the required features to calculate the lateral offset. During this process the perceptual system ensures that the system is robust to identify and map the crack under various environment which has been discussed. Finally, the lateral offset is used to automatically move the robotic manipulator end-effector or tooltip on top of the longitudinal crack.

In the automated mode, the system is in charge of handling the movement of the actuator. The machine vision module in this operational mode extracts features from the tracking view image. Then output of the image processing is used by the controller to move the cutter head on top of the edge crack. The semi-autonomous mode requires the operator to supervise the operation. A comparative functional diagram of these three modes in illustrated in Figure 3-6.

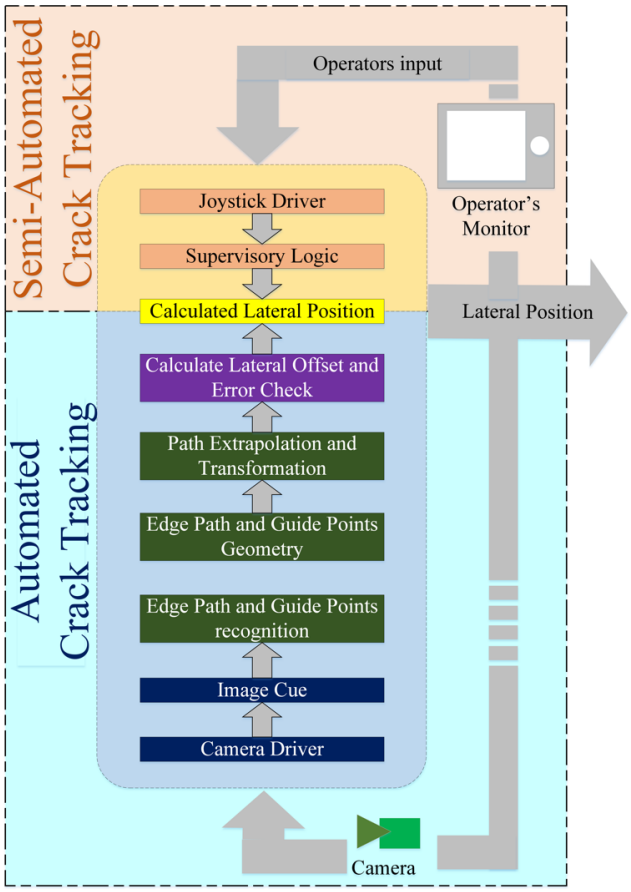


Figure 3-6 Functional Diagram and the Data Flow of the Three Operational Modes of Operations.

3.2.2. Manual mode scheme

One of the sponsors of this research, the California Department of Transportation (Caltrans) initially considered a manual tracking technique for crack sealing/cleaning to be a viable method. In this scenario, a worker in the vehicle monitors a live leading-view video image of the longitudinal joint crack and controls the end-tool's position immediately as shown in Figure 3-7-a. The tool head's location was indicated by an indicator on the live crack video image, and the operator would jog one degree of freedom on a joystick controller to move the end-tool laterally to the desired position. Manual remote-control tracking is, without a doubt, the most elementary control method available. Although it is convenient to deploy the manual tracking system, it is a time-consuming operation that requires a human to maintain constant focus and react quickly. Joint crack video frames travel at a speed of 3 mph or 53 in/sec inside this tiny field of view. At

this rate, even minor changes in joint crack positions are extremely difficult for the human brain/visual system to process and are inconvenient to track with a cursor. The manual camera streaming system was used to follow a longitudinal joint crack at a vehicle speed of three miles per hour. The image was provided to a technical team at Caltrans. After viewing the sample video, they decided to focus on a more automated guidance approach.

3.2.3. Semi-automated and automated guidance schemes

Semi-automated guidance systems integrate automated tracking with supervisory control as shown in Figure 3-7-b. This approach entails the use of both leading- and trailing-view video frames. This method of tracking processes the trailing-view image to identify and track the longitudinal joint crack, whereas the system operator focuses on the oncoming joint crack in the leading-view image. The scheme's fundamental assumptions are that the edge crack is approximately straight. When incongruities in the edge crack are encountered, the supervisory control provides a means of manually navigating the router through these incongruities at a significantly slower speed. The driver is responsible for navigating the vehicle nominally along the longitudinal joint crack in this scheme. The lateral actuator controls the router precisely, preventing contact with the PCC travel lane. During the operation, the system operator selects the most appropriate source of lateral actuator control and alternates between manual and semi-automated tracking control. Thus, whenever inconsistencies with the PCC are detected, the driver will (supposedly) decelerate to allow the system operator to deactivate the semi-automatic guidance system in favor of the manual one. Once the issue has been resolved, the supervisory control will re-engage the automatic system.

Once the system has analyzed the trailing-view image to identify the PCC lane edge and has generated a projected router tracking path, the system operator can switch to automated guiding mode at which point the router will track itself. The trailing-view image is digitally processed using an edge recognition algorithm to correctly identify the PCC lane edge and calculate a projected lateral router position to generate

the automated lateral guidance positioning control. The tool head travels laterally to the required place to follow the joint crack (Figure 3-7-b). The system operator can monitor the automated-guidance system, and bypass it if it will become necessary, by engaging the lateral joystick. Additionally, the system operator would be responsible for initiating routing and initially establishing enough routed joints/cracks for the auto-guidance to lock on. Only the trailing camera's image processing techniques are used to enable automated tracking.

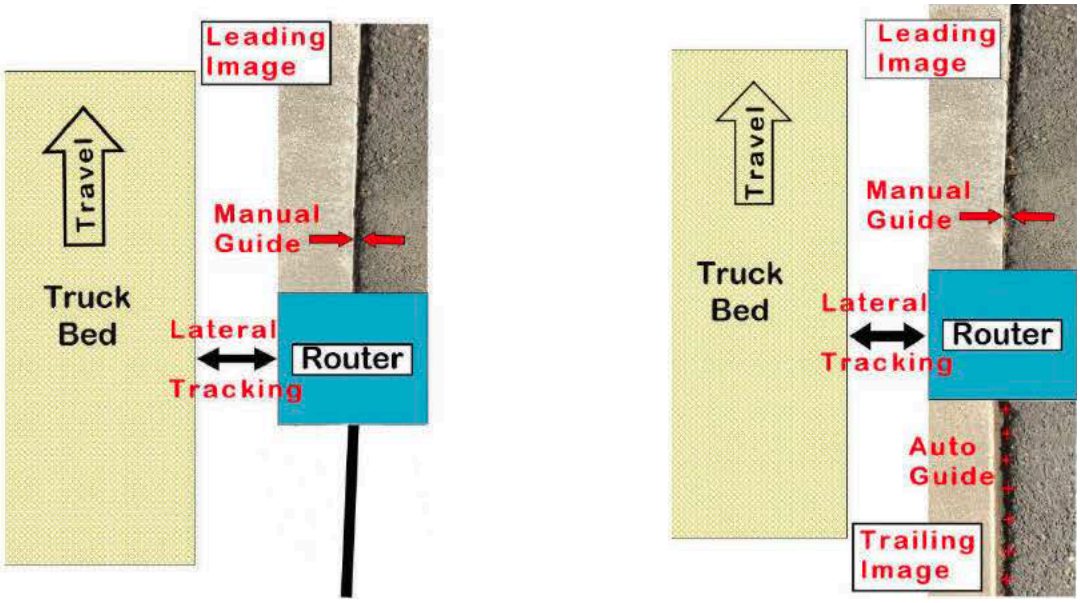


Figure 3-7 Guidance system for operation on longitudinal cracks a) Manual guidance b) semi-automated guidance

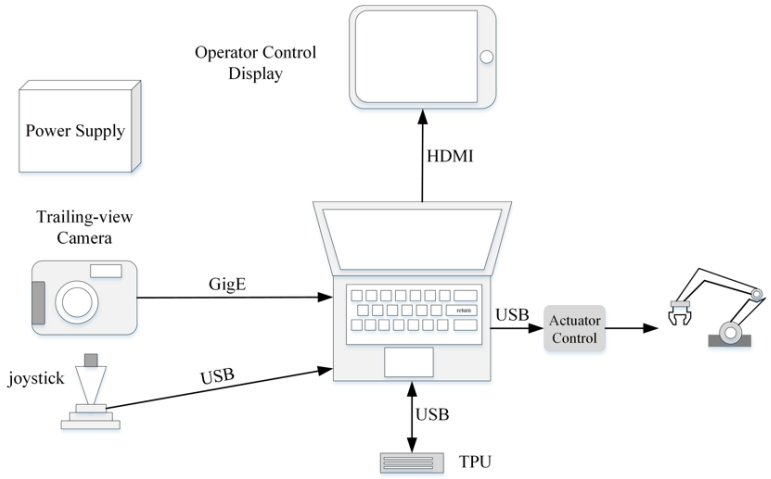
3.2.4. System Software for Tracking

A schematic diagram of the tracking system is shown in Figure 3-8 (a). The Linux-based system schedules events in real time. The laptop communicates with its modules through the USB, HDMI, and GigE ports. The main components are the power supply, the camera, the operator control display, a TPU, and the robot's controller system. Figure 3-8 (b) shows the software modules including:

1. Main application process: Device initialization, shutdown, and user interface handling
2. System control process: Orchestrates the system operation by controlling the other system modules.

3. Video control process: Configures and controls the trailing-view camera device and stores the most recent video frame.
4. Detection control process: Manages the TPU system and transforms and analyzes video frames to estimate the position of the cracks
5. Actuator control process: Manages system actuator motor controller
6. Joystick control process: Manages the joystick system and produces joystick event notifications

a)



b)

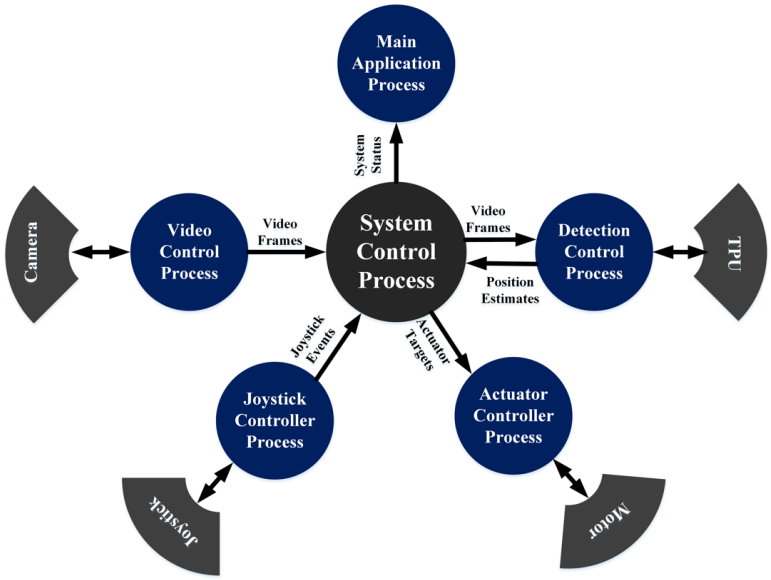


Figure 3-8 a) Operator Supervisor Control Diagram and b) Task planning control architecture of processing and control

The system control process is the core component of the software task planning system. This module integrates all other components and provides the system status for the main application process. The video control process includes the camera driver which obtains and stores the most recent video frame. The detection control process estimates crack position by transforming and analyzing video frames generated by the video control process. The estimation logic (Figure 3-9) makes use of a Google Tensor Processing Unit (TPU) for accelerated inferencing. The joystick controller process is responsible for detecting joystick events and communicating them to the system control process. It is important that the joystick device is serviced several times per second in order to allow for quick response. The actuator control process contains the actuator drivers and manages the actuator motor controller based on input from the system control process.

```
# coding=utf-8

import math
import time
import numpy
from PIL import Image
from PIL import ImageDraw

import crackthresh

class CrackDetect(object):

    COLOR_FILTER_PTS        = '#FFFF00FF'
    COLOR_LINE              = '#0000FFFF'
    COLOR_BBOX_DETECTED     = '#00FF00FF'
    COLOR_BBOX_NOT_DETECTED = '#FF0000FF'

    BBOX_VERT_GROWTH_FACTOR       = 0.4
    BBOX_VERT_GROWTH_THRESHOLD_PX = 100
    DETECTION_MAX_OBJECTS         = 10
    LSTSQ_MIN_PTS                 = 40

    def __init__(self, img_dim, lid_crack, lid_circle, lid_square, lg_mm,
guid_center_x_px, scaling_factor):
        self.img_dim = img_dim
        self.lid_crack = lid_crack
        self.lid_circle = lid_circle
        self.lid_square = lid_square
        self.lg_mm = lg_mm
        self.guid_center_x_px = guid_center_x_px
        self.scaling_factor = scaling_factor
        #
        self.lx_mm = lg_mm - (guid_center_x_px * self.scaling_factor)

    def process_image(self, engine, img_l, cbox_prev, detect_thresh,
simpfilter_thresh, adaptfilter_thresh, adaptfilter_radius, adaptive):
        img_rgb = img_l.convert('RGB')
        draw_rgb = ImageDraw.Draw(img_rgb)
        # perform detection
        ts_begin = time.monotonic()
        cands = engine.DetectWithImage(img_rgb, threshold=detect_thresh,
keep_aspect_ratio=True, relative_coord=False,
top_k=CrackDetect.DETECTION_MAX_OBJECTS)
        infer_time = time.monotonic() - ts_begin
        # get highest-scoring crack_obj
```

```
        crack_obj = None
        crack_det_cnt = 0
        if cands:
            hi_score = None
            for obj in cands:
                if (obj.label_id == self.lid_crack):
                    if ((hi_score is None) or (hi_score < obj.score)):
                        crack_obj = obj
                        hi_score = obj.score
                    crack_det_cnt += 1
        # Determine region to filter
        (x0, y0, x1, y1) = [None] * 4
        if crack_obj is not None:
            # Calculate region by transforming and clipping detected bounding box
            x0 = max(0, int(round(crack_obj.bounding_box[0][0])))
            y0 = max(0, int(round(crack_obj.bounding_box[0][1])))
            x1 = min((self.img_dim - 1), int(round(crack_obj.bounding_box[1][0])))
            y1 = min((self.img_dim - 1), int(round(crack_obj.bounding_box[1][1])))
        else:
            # Estimate region
            prev_height = int((cbox_prev[3] - cbox_prev[1] + 1))
            if (prev_height < CrackDetect.BBOX_VERT_GROWTH_THRESHOLD_PX):
                # base region on previous cbox, adding 40% to the height and
maxing out the width
                delta_y = int(round(prev_height *
CrackDetect.BBOX_VERT_GROWTH_FACTOR / 2))
                x0 = 0
                y0 = max(0, cbox_prev[1] - delta_y)
                x1 = self.img_dim - 1
                y1 = min((self.img_dim - 1), cbox_prev[3] + delta_y)
            else:
                # Use previous cbox as-is
                (x0, y0, x1, y1) = cbox_prev
        # execute filter
        imgbytes = CrackDetect.__crop_to_bytes(img_l, x0, y0, x1, y1)
        if adaptive:
            (x_vals, y_vals, xy_vals) = crackthresh.filter_adaptive(imgbytes, x1-
x0+1, y1-y0+1, x0, y0, adaptfilter_thresh, adaptfilter_radius)
        else:
            (x_vals, y_vals, xy_vals) = crackthresh.filter_simple(imgbytes, x1-
x0+1, y1-y0+1, x0, y0, simpfilter_thresh)
        # draw filter points
        draw_rgb.point(xy_vals, fill=CrackDetect.COLOR_FILTER_PTS)
        # conditionally calculate y_mm
        if (len(xy_vals) > CrackDetect.LSTSQ_MIN_PTS):
            (m, b_px) = CrackDetect.__least_squares(x_vals, y_vals)
            y_mm = -m * self.lx_mm + (b_px - (self.img_dim / 2)) *
self.scaling_factor
            # draw line
            draw_rgb.line((0, b_px, (self.img_dim - 1), (m * (self.img_dim - 1)) +
b_px), fill=CrackDetect.COLOR_LINE)
        else:
            y_mm = None
        # draw cbox
        if crack_obj is not None
            rect_color = CrackDetect.COLOR_BBOX_DETECTED
        else:
            rect_color = CrackDetect.COLOR_BBOX_NOT_DETECTED
        cbox = [x0, y0, x1, y1]
        draw_rgb.rectangle(cbox, outline=rect_color)
        # Return results
        return (y_mm, img_rgb, crack_det_cnt, infer_time, cbox)

    @staticmethod
    def __least_squares x_vals, y_vals):
        a = numpy.vstack([x_vals, numpy.ones(len(x_vals))]).T
        (m, b_px) = numpy.linalg.lstsq(a, y_vals, rcond=None)[0]
        return (m, b_px)

    # convert a cropped region of a PIL mode-L image to a Python bytes object
    @staticmethod
    def __crop_to_bytes(img_l, x0, y0, x1, y1):
        imgarr = numpy.asarray(img_l)
        crop = imgarr[y0:(y1+1), x0:(x1+1)]
        return crop.tobytes(order='C')
```

Figure 3-9 The crack detection source code

3.3. Machine vision crack sensing technique

The semi-automated and the automated tracking component of the guidance system are dependent on creating a method for digitally analyzing the trailing-view image and successfully recognizing the PCC lane edge that runs through the frame. When the joint crack transition surface is clear, the color difference between the predominantly white PCC pavement and the primarily black AC pavement is visible using typical image processing techniques. However, on most PCC roadways, the surface close to the transition joint is frequently hidden by lane striping, old sealant, and pavement markers, introducing inaccuracy and significantly reducing the reliability of standard image recognition approaches.

A new field of intelligent image processing technology is emerging that is well suited for classifying objects inside a digital image [29] Deep learning is a broad phrase that refers to a diverse set of AI machine learning techniques [30] In deep learning, rather than applying a fixed set of predetermined filters that must be programmed, the filters are built internally by the program through learning based on supervised training. Given the ambiguous nature of the trailing-view highway pavement images, a deep learning image processing framework was developed here to deliver the best feasible identification for the crack cleaning/sealing guidance system.

3.3.1. Edge Identification and Edge Path Extrapolation Program

After the edge identification program detected a crack edge path, we constructed a real-time filtering and computer code to extrapolation the next point on the edge crack and determine the lateral end-tool's position. Appendix A contains the transformation equations developed for path extrapolation. Because the crack cleaning guidance method required extrapolating the recognized edge path approximately 30 inches ahead of the longitudinal joint crack, any minor geometric positional errors can become significant. Additionally, the trailing-view camera moves in sync with the guided vehicle, allowing for an unrestricted

view of the joint crack in the image. Because the vehicle's orientation will change somewhat between frames, the joint/crack will constantly shift and rotate slightly within the image. The edge tracking algorithm must adjust for the picture frame inherent movements and calculate an accurate end-tool/router lateral position. This is especially important in crack cleaning when a router is used since the router cutter's inner edge must be kept next to the PCC edge to avoid hitting the stronger AC edge of the shoulder. As a result, for the tracking system to be effective, a method for reducing the inherent errors of this methodology need to be established. Once the edge path has been discovered, the guidance system must translate it into the lateral position of the end-tool/router inside the edge. A router lateral position is calculated by entering a crack path line and guiding point positions in a trailing-view image. The transformation software involves initializing the trailing-view camera and defining critical geometric dimensions between the end-tool/router and the guiding points.

The edge identification and edge path extrapolation programs have four stages (see Figure 3-10). The first stage preprocesses the image data by down-sampling and changing the dimension of the input image as well as converting its color channels. The second stage is the deep convolutional neural network filtering. In this stage, if the network model detects the longitudinal crack, then the detection bounding box is utilized for the next processing stage. If the crack is not detected, an estimated bounding box is created by scaling up the bounding box from the previous detection in order to maximize the probability of crack detection in the subsequent processing stages. The third stage involves threshold filtering which allows for the identification of crack pixels from non-crack pixels using binarization of the image. After the pixel candidates are identified, the final stage utilizes a regression model on the identified crack pixels to find the position and orientation of the edge crack.

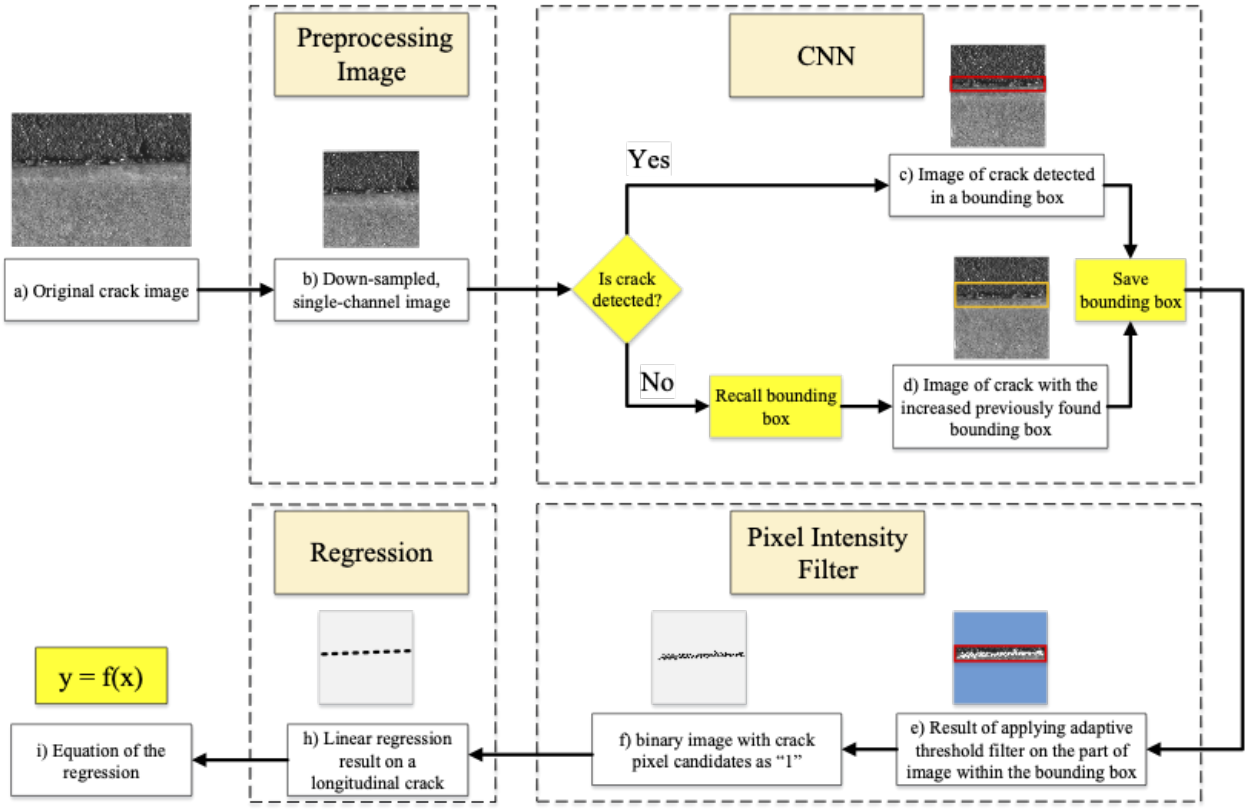


Figure 3-10 Four stages of image processing to identify position and orientation of longitudinal edge cracks

The novel real-time joint crack sensing architecture developed here uses a DCNN method and it is shown in Figure 3-11. This detection technique, which consists of two primary processing stages. First, it takes the original frame from a camera, preprocesses it, and then uses a DCNN-based detection model to find the longitudinal crack within a bounding box. Second, it binarizes the identified crack picture in the bounding box to map the crack using an adaptive threshold approach. The output is then sent to the task planning system for real-time manipulator actions.

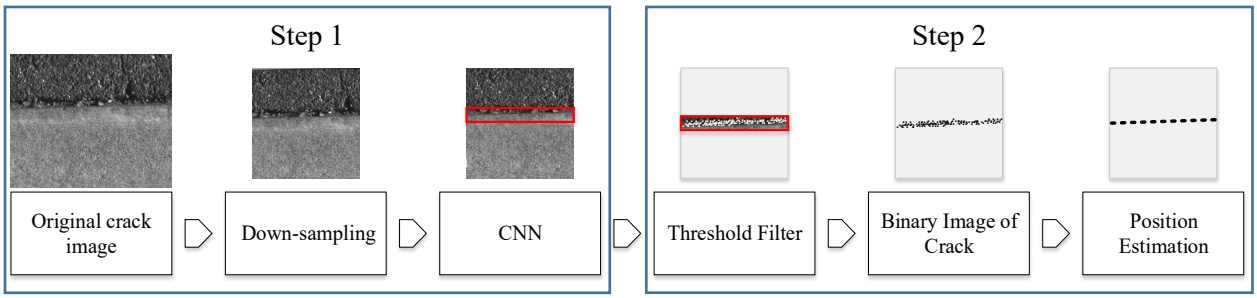


Figure 3-11 Details of hybrid image processing technique: 1: down-sampling 2: Applying CNN 3: applying threshold 4: Binarizing the image 5: estimate the position and direction of the longitudinal crack

3.3.2. **Statistical line fitting and coordination transformation**

As discussed earlier, for the robot arm to move and track the longitudinal edge crack, the distance between the robot's end effector and the longitudinal crack must be calculated. This process requires two more steps after finding the crack pixel candidates through the previous two filters. The first step uses statistical line fitting to the crack pixels as illustrated in Figure 3-12-a and the second step uses a coordination transformation moving the data from the camera frame to the world frame as depicted in Figure 3-12-b.

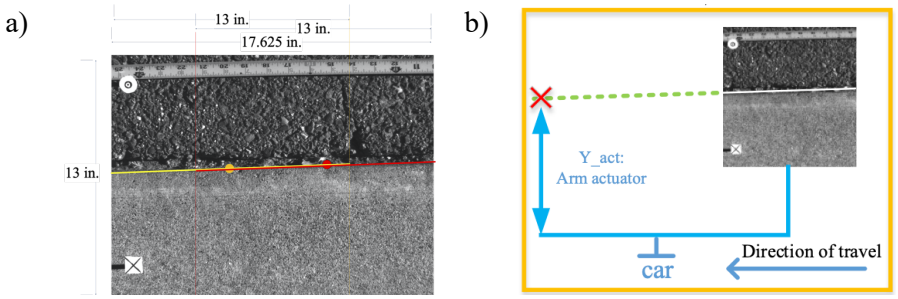


Figure 3-12 a) Longitudinal crack frame dimensions b) Show the extrapolation of the crack line

Line fitting is the process of calculating a line that would best fit a set of data points [31]. The statistical line fitting to the crack pixel candidates fits a straight line on top of the straight edge of the pavement as can be seen in Figure 3-13. The equation of the fitted line will provide the relative orientation and relative position of the longitudinal edge crack.

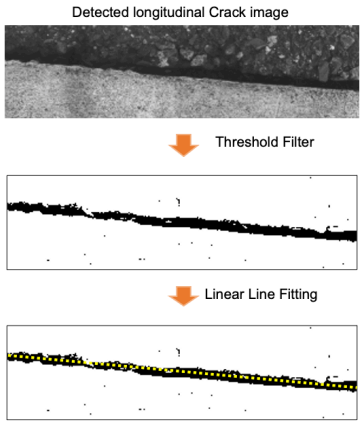


Figure 3-13 Crack mapping and linear line fitting to the detected longitudinal crack

There are three main types of statistical line fitting which is mentioned in [32]:

- OLS (Ordinary Least Squares)
- LNS (Least Normal Squares)
- OC (Line of Organic Correlation)

All the above-mentioned methods pass through the centroid of the data set but may differ in the way they determine the slop [32]. In our case, since we only need to estimate the longitudinal crack orientation and direction, we can use the simple OLS method also known as the regression estimating method.

The linear equation of the fitted line is:

$$\bar{y} = b\bar{x} + a \qquad \text{Equation 3-1}$$

where the parameters $\bar{x}$ and $\bar{y}$ are defined as:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

with (x_i, y_i) are the coordinates of all the data points. The Least Square formulation is to minimize the following equation:

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad \text{Equation 3-2}$$

$$\hat{y}_i = b\hat{x}_i + a \qquad \text{Equation 3-3}$$

The optimal values for a and b are obtained from:

$$b = r\frac{S_y}{S_x} \qquad \text{Equation 3-4}$$

Where r, S_y, and $S_x$are:

$$S_x^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2 \qquad S_y^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2 \qquad r = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{S_x S_y}$$

The parameter a is found from:

$$a = \bar{y} - b\bar{x} \qquad \text{Equation 3-5}$$

Now having the values for a and b, we can estimate the position and orientation of the longitudinal crack.

3.3.3. **Kinematic Registration**

The robotic arm with its end-tool/applicator is installed on a moving vehicle while the edge cracks to be filled or routed for cleaning are on the roadway. This means that the position of the crack relatively to the end-tool must be estimated in real-time using the camera images requiring a kinematic registration. Kinematic registration involves establishing the coordinate transformation associated with establishing the correspondence between the roadway and the coordinate system of the robotic system in the moving vehicle. The position of the crack is available in the image coordinate system in pixels. The pixel distance then needs to be converted from the image coordinate system to regular distances in the robot tool coordinate system.

Different coordinate systems for a maintenance truck with a robotic tool for crack sealing/routing is depicted in the Figure 3-14 application. In this Figure the subscript “w” is used to indicate the world coordinate system, “c” is used for the car or vehicle coordinate system, “a” for the actuator and “i” for the image coordinate system. In the figure, the yellow arrow shows the direction of the travel. The world coordinate system includes the roadway with the longitudinal edge crack. The Car or the vehicle coordinate system is attached to the vehicle and it includes the camera mount. The 2D (2-Dimensional) Image coordinate system is set to the image recordings. The Actuator coordinate system is attached the robotic arm.

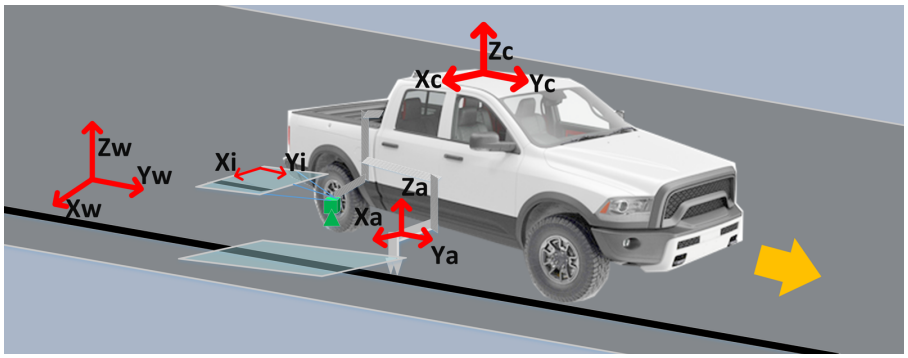


Figure 3-14 Coordinate system representation of crack tracking system including a truck with the attached linear robotic actuator.

The coordinate transformation chain includes the transformation from the World coordinate to the Image coordinate, from Image coordinate to the Vehicle coordinate and from vehicle coordinate to the Actuator coordinate as follows:

$$C_A^W = C_I^W C_C^I C_A^C \qquad \text{Equation 3-6}$$

The following notation is used in the above equation:

$$C_{to\ a\ frame}^{from\ a\ frame}$$

In this application, internal sensors are used in the vehicle and the robot actuator so that all the values are relative to the position of the vehicle therefore the equation can be simplified as follows:

$$C_A^I = C_C^I C_A^C \qquad \text{Equation 3-7}$$

The coordinate transformation from the image to the vehicle is denoted by C_C^I which includes scaling and translation. The coordinate transformation from the vehicle to the actuator is denoted by C_A^C and includes translation and if the end effector is subject to vibration, then roll, pitch, and yaw must also be included.

In our experimental prototype, the shooting angle of the camera is downward and perpendicular to the pavement surface. To ensure perpendicularity, we designed an initializing fixture. The fixture is attached to the actuator robotic arm (Figure 3-15) and contains a rectangle of guide points ($A_{GP}B_{GP}C_{GP}D_{GP}$). If the

guide points compose a perfect rectangle in the camera image frame (i.e., $G_p = G_{p'}$ and $H_p = H_{p'}$), then the camera angle is normal to the initializing fixture.

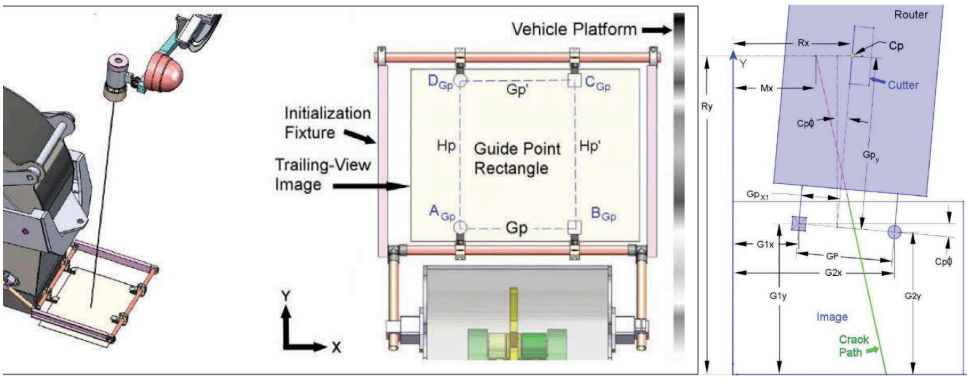


Figure 3-15 Guidance frame design for the crack cleaner system

3.4. Data Collection and Model Training for Machine Learning Algorithm

The most self-evident constraint in model training for machine learning application is a lack of data. Erroneous model input results in incorrect outcomes. As a result, we attempted to collect correct and relevant data to train the model for our machine learning algorithm used for crack detection. For this, we collected data for joint/crack detection in two separate settings: one with a truck on a highway in California and second at a test site at UC, Davis. We mounted a set of rails on the side of a service vehicle and attached it to its bed. We then installed two cameras on the rail: one for the trailing view and another for the heading view. The trailing-view camera was placed directly on top of the crack with the lens pointing downward delivering image data for the crack detection module. Heading-view cameras were mounted in strategic positions to view the horizon along the longitudinal crack and assist the driver and operator in directing the vehicle forward in a straight line. Both cameras were configured to begin and stop recording remotely to help ensure the safety of both operators and equipment.

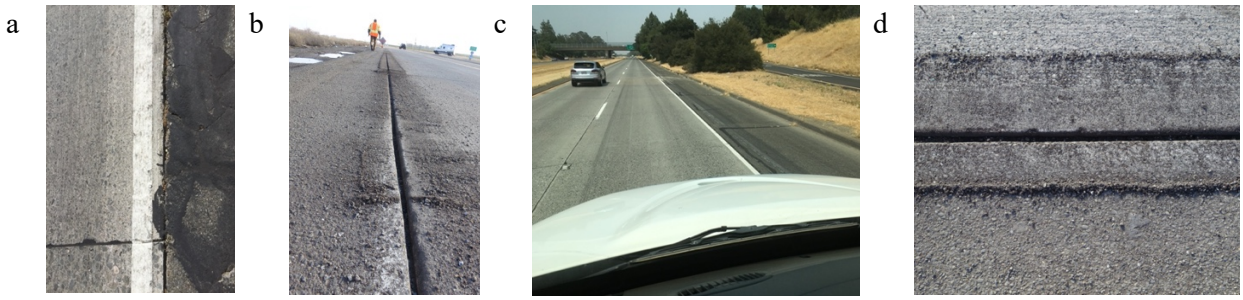


Figure 3-16 First round of data collection for training the network from California highways a) Regular crack before routing/cleaning crack, b) Longitudinal crack after routing, and c) Truck moving and collecting data d) Example of the collected routed crack image

The second round of data collection took place at a site on the UC Davis campus with pavement that bore a resemblance to the AC/PCC pavement joint cracks seen in Figure 3-16. The data was collected using the testbed (Figure 3-17) described in Section 3.5. The cameras were programmed to capture frames at 15 frames per second. A subset (150) of the collected frames were selected for random and split-sample training, testing, and validation, with the split ratio of 60, 20, and 20.

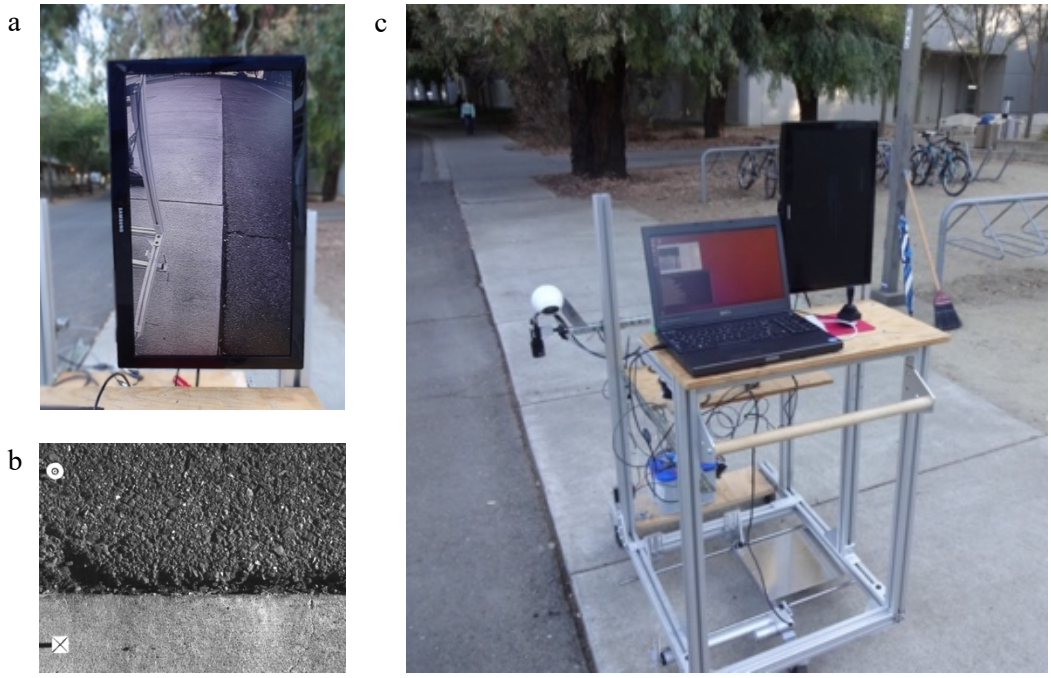


Figure 3-17 Second round of laboratory data collection setup at UC Davis: a) Leading-view camera, b) Sample of collected image of longitudinal crack with guidance labels, and c) Testbed view during image collection

We added guiding frame labels throughout the data collecting process as seen in Figure 3-17-b. These guiding frame labels are used as feedback for the guidance control system. These labels are affixed to a frame that is connected directly to the end effector. The end effector's location may be calculated with a high degree of precision using these guiding frames, especially when there are disturbances such as in crack cleaning and routing. Because the robotic arm shakes when the cutter blades come into contact with the ground, the cutter blade would otherwise deviate from its planned route. Having a feedback system to provide precise information on the position or forces applied to the end-effector is advantageous in controlling the system.

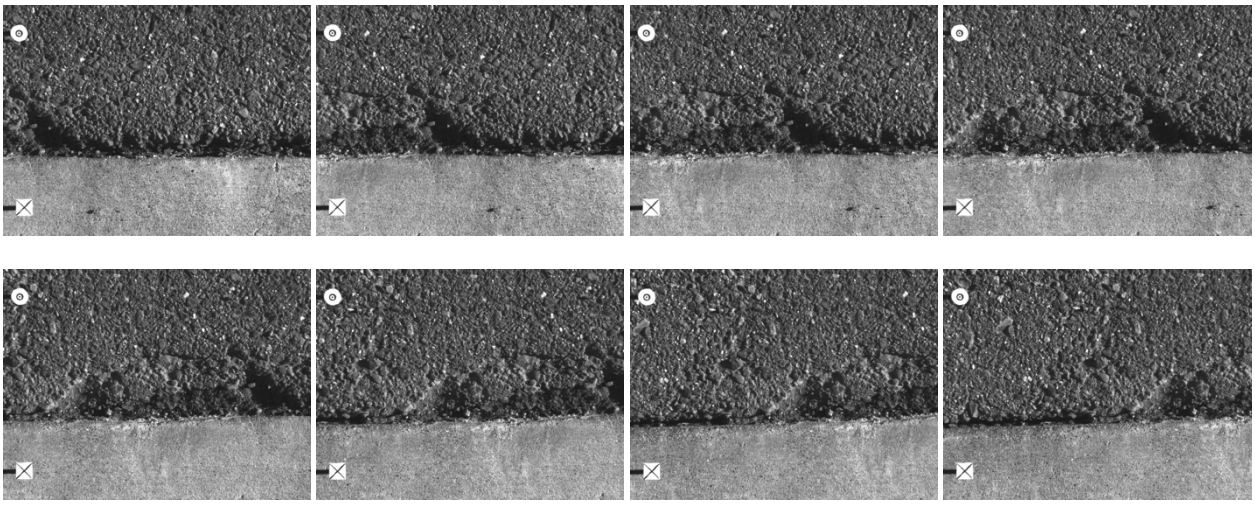


Figure 3-18 Collected data with laboratory setup with guidance frame labels

3.4.1. CNN based Filter: Model selection and training

Transfer learning is a robust framework in deep learning in which pre-trained models are used as the starting point for computer vision and natural language processing tasks.

Several models are selected, trained, and tested through the transfer learning technique for the crack detection task. Numerous criteria were examined in making these model selections, including:

- High accuracy or high mAP (mean Average Precision) to distinguish between cracks and non-cracks
- Rapid inference speed: to locate the crack in the shortest amount of time possible
- Real-time application: must be applicable in a robotic system for real-time processes
- Being quantized: to offer the quantized output for the tensor processing unit's vector calculation

Considering the four abovementioned criteria, we selected various models from the Model Zoo collection. Model Zoo is a platform for developing and deploying machine learning models and emphasizes transfer learning applications [33]. After experimenting with various models, we determined that SSD-mobilenet-v2-quantized-coco best met the aforementioned requirements. It had the best model precision with the specs of 22 mAP and 29 milliseconds inference speed for a single image. This model generates a bounding box around the longitudinal edge crack. Finally, we used a 1070 TI GeForce GTX graphics card to train the model. The training result is shown in Figure 3-19.

Figure 3-19-a shows the learning rate curve of crack model training using 120,000 epochs for training. The number of epochs is a hyper-parameter that specifies how many times the learning algorithm covers the training dataset. Each epoch is a chance for each sample in the training dataset to change the internal model parameters. The learning rate plot indicates that the mean average precision (mAP) is about 86%. We could have trained it further, but our experience with the first highway dataset indicated that we would fall into the overfitting problem; therefore, we discontinued training at this stage. Overfitting is a term that refers to a model that is too precise in predicting the training data resulting in capturing noise or other random variations in the training data degrading the model.

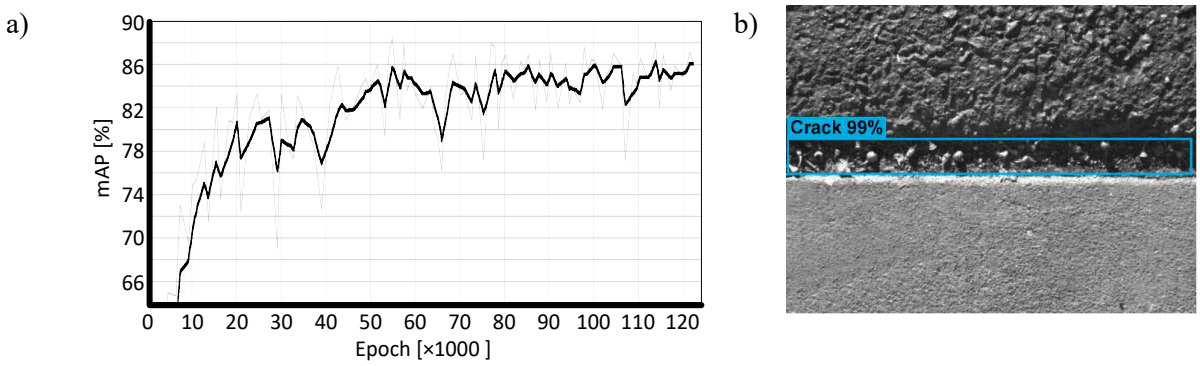


Figure 3-19 Results of training the CNN-based model: quantized Mobilnet V2 a) Learning rate and b) detection trial that crack detected with 99% accuracy

3.4.2. Threshold filtering

As part of filtering, Simple Threshold (ST) and Adaptive Threshold (AT) filters were implemented to map the crack from the identified crack bounding box. If the pixel value is lower than the global threshold value, the ST filter considers it as a crack pixel, and if they are more than the global threshold value, considers it as a non-crack pixel. In the AT approach, the threshold value for each pixel is calculated locally. Our experiments indicated that the AT outperformed the ST. An example result comparing the two thresholding methods on an edge crack is depicted in Figure 3-20.

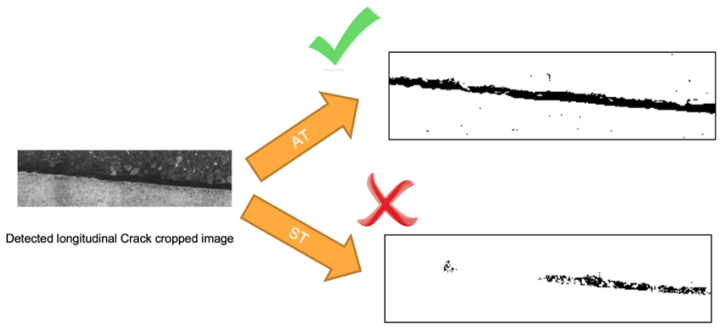


Figure 3-20 Results comparing simple versus adaptive threshold on a longitudinal crack; the top model with AT outperforms the ST method

3.4.3. Filter parameter tuning

The crack mapping outcomes produced by threshold filtering are not necessarily optimal. We optimized these filtering parameters by modifying the confusion matrix to modify our crack detection technique. A confusion matrix is a tool for evaluating the performance of a classification model. It compares the ground truth (actual values) and the predicted values for the target variable generated by the model. The four quadrants of a confusion matrix are:

- True Positive (TP) is an outcome that occurs when the model predicts the positive class correctly.
- True Negative (TN) is an outcome that occurs when the model predicts the negative class correctly.
- False Positive (FP) is an outcome that occurs when the model forecasts the positive class wrongly.
- False Negative (FN) is an outcome that occurs when the model forecasts the negative class wrongly.

Figure 3-19 depicts the parameters of the confusion matrix as applied to a crack image in its bounding box.

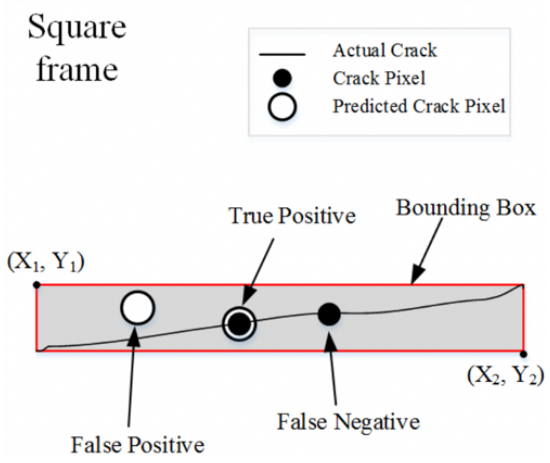


Figure 3-21 Representation of TP, TN, FP, and FN for distress detection and calculating accuracy

The parameters of the confusion matrix, TP, TN, FP, and FN illustrated in Figure 3-21 are interpreted as follows. TP will correspond to the correct identification of the longitudinal crack. TN will correspond to the correct identification of the intact road pavement. FP are incorrect predictions of the longitudinal crack class, while FN are incorrect predictions of the intact pavement surface.

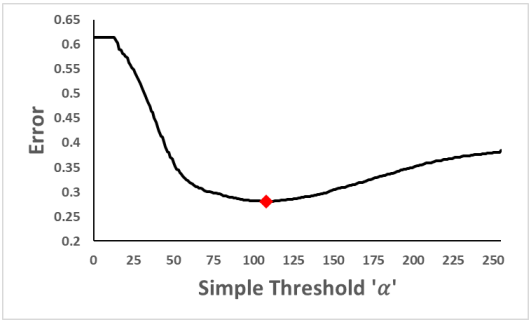
In order to improve the accuracy of our crack detection method, we considered the image reference to be the portion of the captured frame within the bounding box rather than the entire captured frame. In other words, whenever a crack was identified, we stored its bounding box information and then computed its accuracy concerning its bounding box. This modification of the method results in the bounding box size (i.e., its height and width) to change with each detection to ensure that the filter parameters are optimized. The errors in thresholding are calculated using the following equations:

$$Error_{BBox} = \frac{FP + FN}{All\ pixels\ in\ BBox} = \frac{FP + FN}{(X_2 - X_1 + 1)(Y_2 - Y_1 + 1)} \quad \text{Equation 3-8}$$

$$Error = \frac{\sum_{i=1:n} Error_{BBox}\_i}{n} \quad \text{Equation 3-9}$$

In these equations, X_1, X_2, Y_1, and Y_2 are the coordinates of the bounding box (*BBox*), and n is the frame number. The first equation calculates the error value for a single frame with a detected crack, and the second equation accumulates the error for n frames. The two graphs in Figure 3-22a and Figure 3-22b illustrate the outcomes of the optimization of the thresholding parameters. The same input and error measure were used to generate both graphs. Comparing Figure 3-22a and Figure 3-22b, the minimal error value obtained for the ST filter was 0.28, whereas the value obtained for adaptive thresholding was 0.24. A lower error score for the filter indicates better crack mapping.

a)



b)

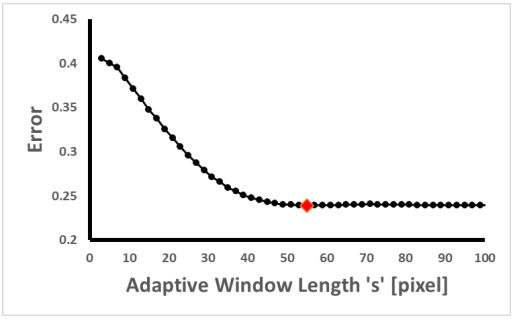


Figure 3-22 Parameter optimization vs. the error: (a) The ideal ST parameter value is at 108, with an error of 0.28 and (b) The ideal threshold value for adaptive thresholding is s= 55 t = -0.07, with an error of 0.24.

ST filtering is a practical approach with straightforward implementation, and its parameters are easy to tune. The results, however, are not resilient to changes in lighting conditions or backdrop texturing. On the other hand, the adaptive thresholding approach is more challenging to apply and to fine-tune its settings. AT is more resilient than ST when it comes to producing the crack map.

3.4.4. Detection results

Sample experimental results from many tests that we performed with our crack detection system is shown in Figure 3-23. We tested in a variety of challenging scenarios such as rainy, shadowy, cluttered, and dark environments. The crack image no. 1 in Figure 3-23 shows the normal condition longitudinal crack in an ordinary situation. The crack image no. 2 illustrates the crack in a damp environment that may occur in rainy conditions. The light reflection on the wet surface of the road causes a signal error in the detection process; but the detection is robust enough to overcome this issue. The crack images no. 4 and no. 5 are in shadowy and darker settings. The crack image no. 4 presents the challenge of dealing with contrast between the dim light and shadows. The effectiveness of our crack sensing method can also be seen in the dark setting of image no. 5. The crack detection method developed is using the little clues in the image to detect the crack. The crack images on the second row from image no. 6 to image no. 10 are all scenarios where the longitudinal crack is covered by vegetation. The crack image no. 6 shows the partially cluttered scenario where half of the crack image is covered by vegetation. The crack image no. 7 shows a heavy cluttered situation in which the trace of the crack and its edges are small, but the algorithm can still detect the crack. Crack image no. 8 shows a garden hose that we had placed in the vicinity of the longitudinal crack. In this scenario, we are trying to deceive the model cluttering the space with an object that looks similar in an image to a roadway crack. As can be seen in crack image no. 9, although there are objects crossing the longitudinal crack, the crack and its orientation are very well estimated by our crack detection system. Crack image no. 10 is an example of failing detection. In the case of a detection failure, a backup detection is used which utilizes the previous image that is still in the memory. This backup detection would work

even if there are up to five consecutive detection failures. If the number of consecutive failures pass the number five, then the system notifies the operators that there was a detection failure.

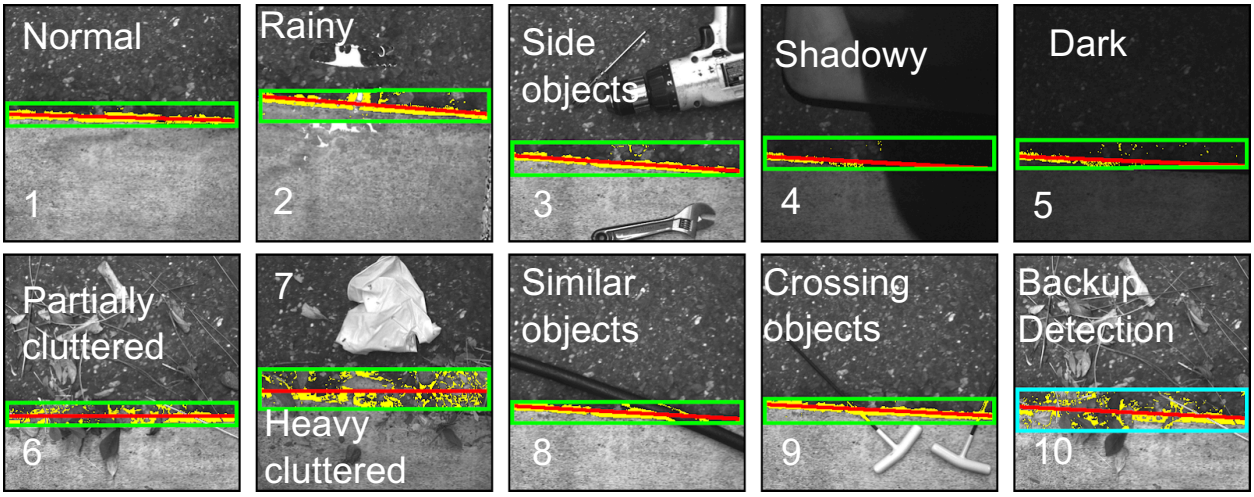


Figure 3-23: Results of the MV technique on challenging scenarios such as rainy, shadowy, cluttered, and dark

3.5. Proof-of-concept laboratory testbed development and demonstration

In order to demonstrate the concepts developed in this dissertation, we utilized a simple experimental setup to demonstrate the operation of the system. The experimental prototype system developed uses a cart that we pull around to model a moving vehicle and a linear actuator moving a pointer as an end effector (modeling a simple robot arm) laterally based on the signals developed from the crack detection system. The camera system with its computer set up containing the detection algorithm were all mounted in the cart as shown in Figure 3-24. The details of all the sub-systems are described in the sections that follow.

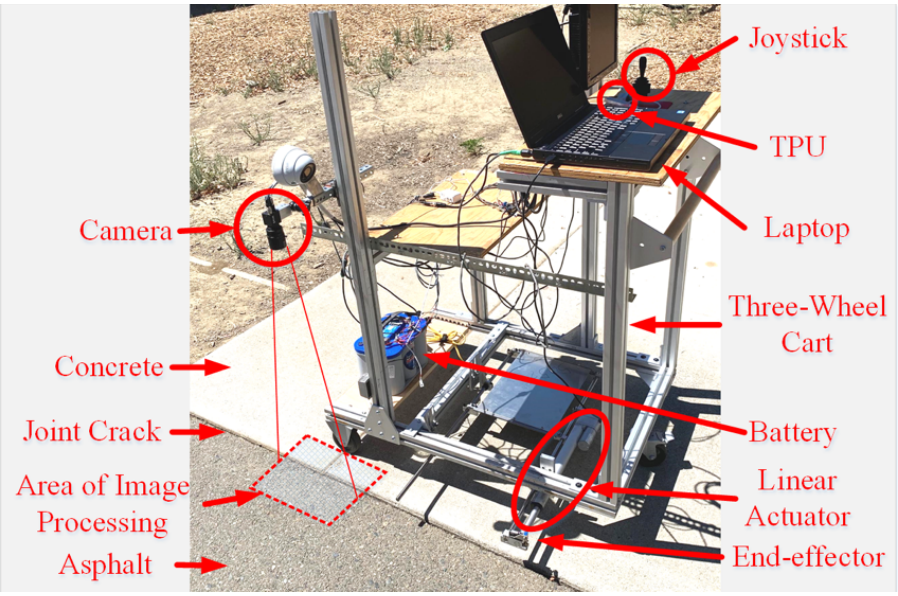


Figure 3-24 Components of experimental system including a self-contained cart with camera, laptop, joystick, battery, actuator, and laser

3.5.1. Camera setup for data collection

The camera was installed in the middle of the lateral positioning travel distance, at the height of approximately 40" and perpendicular to the pavement surface as depicted in Figure 3 25. This installation provides good functionality of the trailing-view video image. In the actual system the trailing-image camera is attached to the end of the tandem camera mounting which will be permanently connected to the truck body, and hence unaffected by the lateral motion of the tool head. The guidance tracking system houses the high-resolution trailing-view camera in a protective aluminum weather-tight enclosure. The cover protects it from physical damage while in this exposed position.

The crack sealing/routing or cleaning equipment is planned to work at vehicle low moving speeds of up to 3 mph in moving traffic lane closures. As a result, the trailing-view camera must be capable of recording a high-quality image perpendicular to the pavement surface while the vehicle is traveling slowly. A camera with a fixed aperture shutter speed will typically smear the pixels at a speed of three miles per hour, resulting in a fuzzy image. A specialized machine vision camera with a configurable high-speed shutter is required

to address image blurriness for this application. The camera used to capture the trailing-view image, in our experimental system, is a reasonably inexpensive machine vision FLIR Blackfly 52 frames/second CCD camera connected through a fast GigE connection to the image processing computer. The Blackfly features a software-controllable high-speed shutter and a robust open-source driver software library to aid in integrating custom crack repairing machines. The Blackfly camera features a shutter speed of 0.019 milliseconds, which in preliminary testing, virtually eliminated pixel blurriness at the intended 3 mph. Because the digital imaging application processes only one image at a time, it only needs to access the most recent high-resolution camera image available from a computer buffer when required. As a result, the trailing-view camera is set up to save a continuous stream of still images at a defined rate, rather than a video format, which would require additional data compression and would degrade image resolution.

The control system guides the tool head within a half-inch PCC roadway edge to ensure effective operation. The picture resolution of the tracking system's trailing-view camera must be at least an order of magnitude greater than half the positioning tolerance to achieve the required level of accuracy. As a result, the trailing-view camera should have a minimum resolution of 0.015" for a 12" to 16" image width. This equates to a camera CCD line resolution of around 1,000 pixels within the range of commonly used machine vision cameras. The FLIR Blackfly camera chosen has a 1/3 CCD format and a resolution of 1280 × 960 pixels.

3.5.2. Testing inside the lab

To verify the crack detection system system's effectiveness, we began testing it in a laboratory setting. To do this, we photographed several longitudinal cracks from a highway in California and then printed them on a long A3 paper and stacked them together to resemble the actual longitudinal crack. These experiments enabled us to fine-tune all the tunable parameters, such as the image processing filters, the PID control parameters, and kinematic relation parameters. Additionally, we were able to resolve a few software communication issues, such as the checksum errors. Later, we discovered that the associated USB port was

malfunctioning and used another port to troubleshoot it. Although some of these faults were minor, they were sufficient to cause the system to fail initially during the field test. The experimental set up for our laboratory testing is depicted in Figure 3-25. The printed photographs of the longitudinal cracks used is shown in Figure 3-25a while the close-up camera location is better depicted in Figure 3-25b.

a) b)

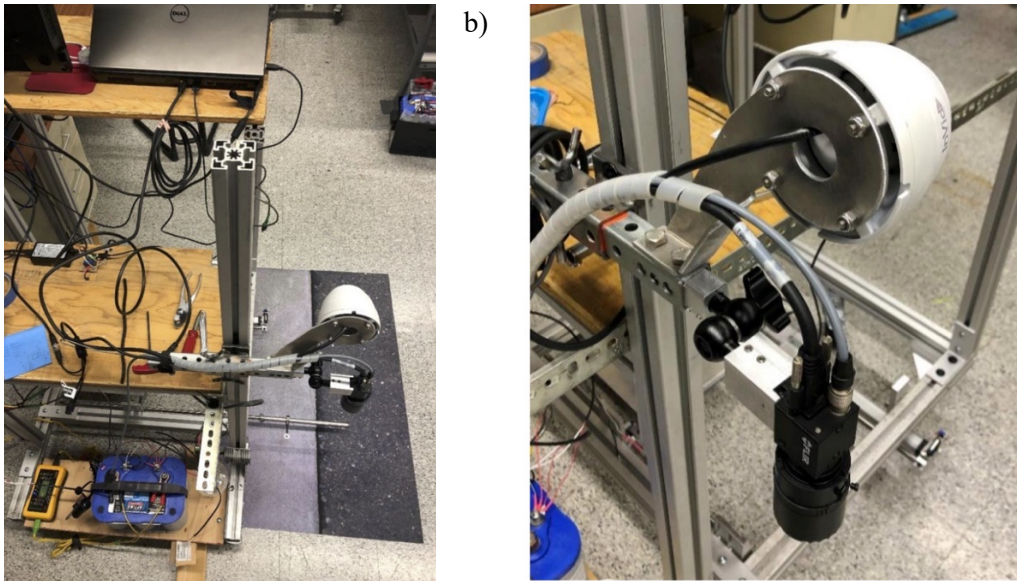


Figure 3-25 The experimental test setup in the lab with paper printed version of AC joint a) During the testing process and b) Trailing-view and heading-view cameras mounted on the cart

3.5.3. Testing on UC Davis roadways

The test cart was relocated to a test site road at UC Davis for field testing to evaluate the effectiveness of the crack detection system in a more realistic setting as illustrated below in Figure 3-26. This figure depicts the cart being rolled down a short path of a PCC asphalt pavement at speeds of up to 3 mph while being pushed by hand. A laser pointer is attached to the lateral actuator's end to indicate the location of the end tool applicator. We undertook ten test runs with the goal of maintaining the commanded position value of the laser, which represents the point of contact between the crack and the cutter blade, to within ½ inch (1.27 cm).

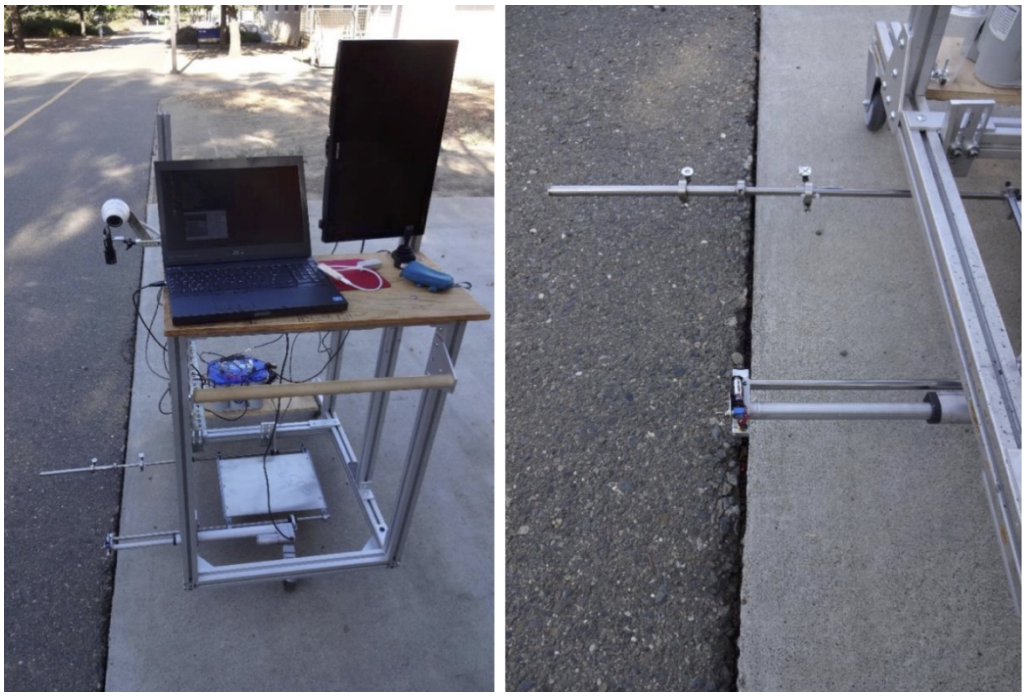


Figure 3-26 Field test: a) Integrated guidance test cart and b) Close-up actuator with the laser pointer for evaluation

We classified each test into three categories according to the lateral magnitude of the motion: low range between 5 and 10 cm, medium range between 10 and 15 cm, and high range between 15 and 20 cm which was the max. The test runs 4 and 9 had a low magnitude range. Low magnitude is an ideal situation as the vehicle moves very precisely. Although this is a desirable tracking test, practically speaking, it is hard to achieve.

Figure 3-27 illustrates a representative sample of all test instances. As depicted in the image, the top row is the fourth test run, demonstrating low range movement. The middle row, as seen in the figure, is the eighth test run, which shows medium-range motion magnitude. Finally, we have the fifth test run in the bottom row, illustrating the high magnitude cart motion.

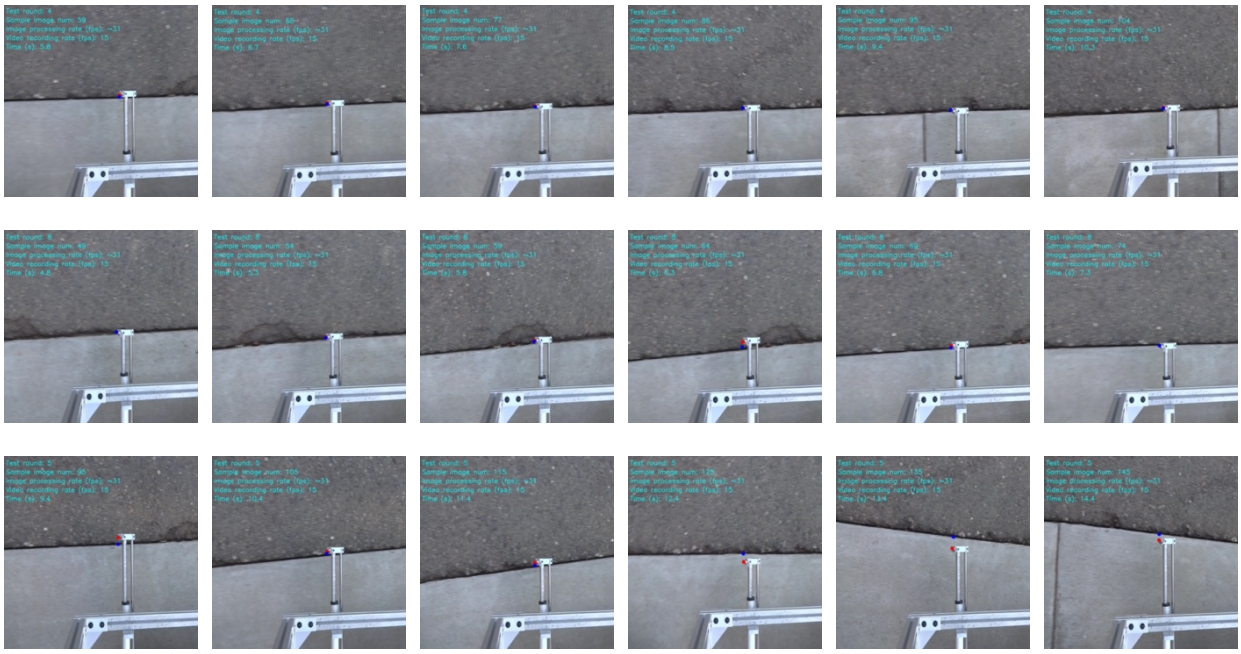


Figure 3-27 Test field at UC Davis. First row is the low amplitude maneuvering, second row is an example of medium amplitude maneuvering, and last row is the example of high amplitude maneuvering

3.5.4. Results and Discussion

Test runs 1, 3, 6, 8, and 10 are with the medium range motion magnitude (10-15 cm). We have collected more from this range since our highway maintenance experts would think that this is range that the driver will likely move the vehicle. The test runs 4 and 9 are at the low range motion magnitude (5-10 cm). This is an ideal case where the truck moves very accurately. The test runs 2 and 5 are in the range of high motion magnitude (15-20 cm). In conducting this range, we accelerated and decelerated the cart to model the worse scenario in terms of vehicle driving. The results of the ten runs are depicted in the images in Figure 3-28.

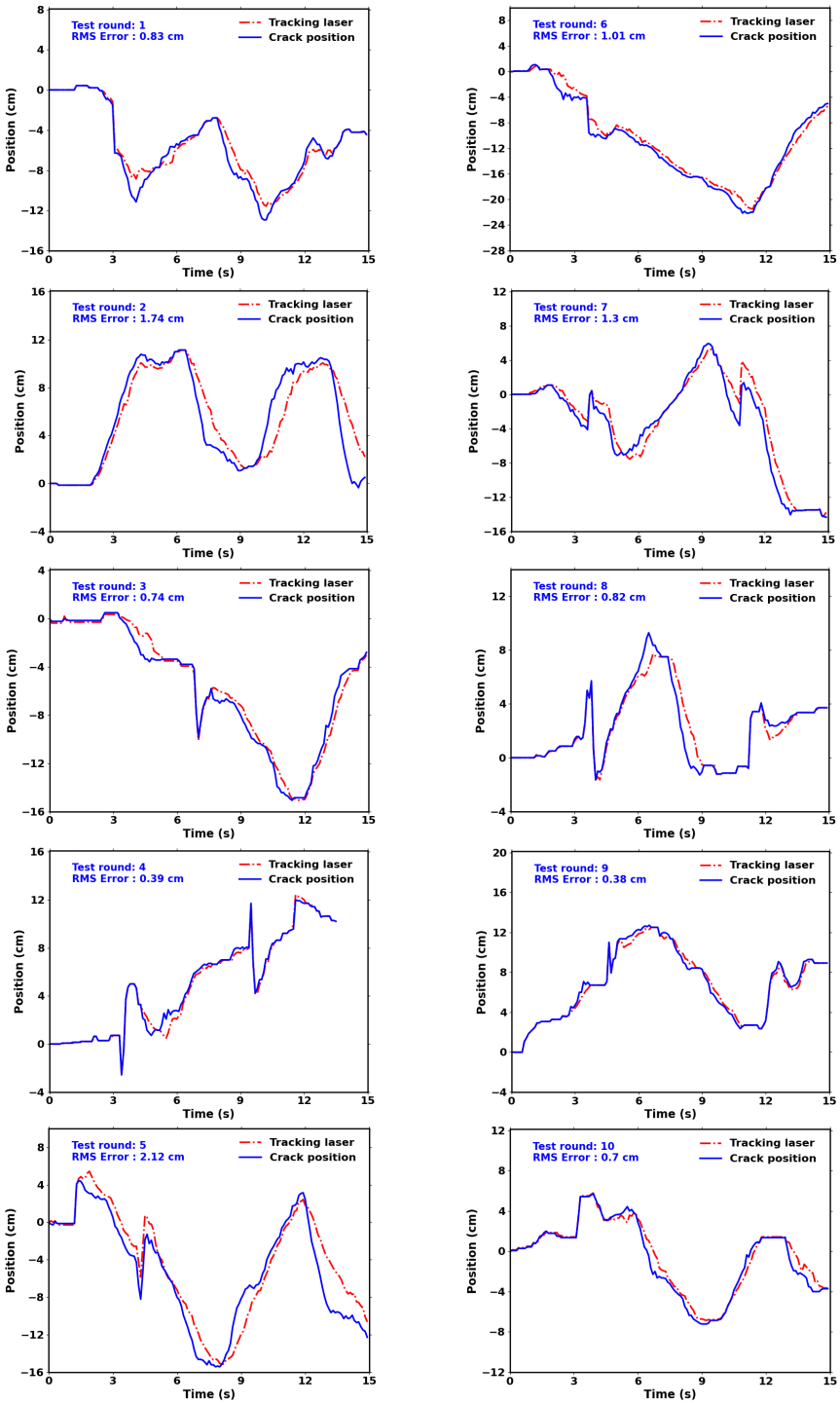


Figure 3-28 Validation Results of the experimental system for ten test rounds.

The average root-mean-square error (RMSE) of the low range movement was around 0.39 cm. The average RMSE for medium-range motion amplitude was at 0.9 cm. Finally, the average RMSE for the high range

cart motion was 1.93 cm. The results of the RMSE are plotted in Figure 3-29. The data for each test are summarized in table 3-1. It is clear from these that the average RMSE value in centimeters for the low and medium motion magnitude test runs was less than 1.00 cm, which is less than our goal of ½ inch (1.27 cm).

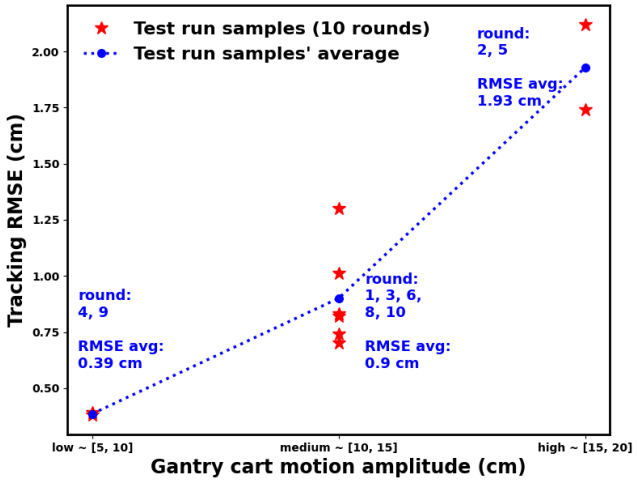


Figure 3-29 The RMSE for different motion amplitudes.

Table 3-1 Result table for longitudinal crack tracking capability of the crack detection system

test round	**RMSE (cm)**	**difficulty level**	**difficulty level amplitude**
1	0.83	2	medium ~ [10, 15]
2	1.74	3	high ~ [15, 20]
3	0.74	2	medium ~ [10, 15]
4	0.39	1	low ~ [5, 10]
5	2.12	3	high ~ [15, 20]
6	1.01	2	medium ~ [10, 15]
7	1.30	2	medium ~ [10, 15]
8	0.82	2	medium ~ [10, 15]
9	0.38	1	low ~ [5, 10]
10	0.70	2	medium ~ [10, 15]

CHAPTER 4

4. Simulation of Sensor-Directed Robotic Task Planning for Space Manufacturing Applications

4.1. Introduction

Extensive logistical support is required for extended human activities in space, both in Low Earth Orbit (LEO) and beyond [34]–[36]. Spare parts, equipment, consumables, and other necessary components for the International Orbit Station (ISS) are periodically transported into space from Earth by a varied fleet of government and private resupply ships [34]. Longer mission durations increase the likelihood of failures, and as a result, the logistical mass required to compensate for them. Additionally, the absence of immediate options that would safely return a crew to Earth in an emergency on the ISS raises the critical importance of different system failures [37]. Logistics, in aggregate, are a considerable operational expenditure for the International Space Station and a significant impediment to safe, inexpensive human exploration beyond the station [34], [37], [38]. The National Aeronautics and Space Administration (NASA) of the United States highlighted the need for in-space assembly, fabrication, and repair (ISAFR) in their Space Technology Roadmaps under Technology Area-12 Manufacturing [39]. The ability to conduct manufacturing activities in space significantly improves mission capabilities. Manufacturing in space will enable missions beyond Low Earth Orbit (LEO) to be extended. Crews would be able to become more self-sufficient and would be less dependent on prefabricated products [40].

ZBLAN fiber optic cable is an example of a product better manufactured in space than on Earth [41]. In early 2018, zero-gravity manufacturing was tried on the ISS, and it was discovered that the tiny crystals that frequently form in fiber produced under Earth's gravity were missing [42]. ZBLAN, a high-grade, nearly perfect fiber optic cable made in space by the Made in the Space business illustrated how microgravity manufacturing might generate superior goods for a range of commercial space and Earth uses [41].

Tele-robotics has been used in space and sectors, such as medicine, for a long time [41]. For example, when a space probe is launched to Mars or the outer solar system, a human technician on Earth monitors its

progress and commands the spacecraft to modify its course in space or direct a rover over an intriguing feature on the planet's surface [41].

Different planning technologies have a variety of uses in the space exploration industry. Deep Space 1 (DS1) was the first flight of NASA's new millennium program launched from Cape Canaveral on October 24, 1998 [21]. It was chartered to test, in space, new technology developed by NASA. On December 18, 2001, the spacecraft was retired after successfully completing the DS1 mission by encountering Comet Borrelly and providing the highest-quality photographs and other scientific data ever returned from a comet [21]. DS1 demonstrated the viability of 12 innovative technologies, including novel mechanical, solar, and control systems, as well as novel software components. Between May 17 and 21, 1999, the Autonomous RA (Remote Agent) software system, which incorporated automatic planning methods, was successfully tested during an experiment on-board DS1 [21], [43].

4.1.1. Space maintenance strategies

According to [44], maintenance strategies, which are tabulated in Table 4-1, are classified as follows: Modes, Types, Levels, and Imagery & Fit checks. OBM ensures station survival, crew protection, mission performance, and cargo operations support while maintaining a reasonable degree of system operability [44].

There are three main maintenance categories in ISS: External Vehicular Robotics (EVR), External Vehicular Activities (EVA), and Internal Vehicular Activities (IVA). IVAs are the most time-consuming activities to do at the ISS. According to [44], more than two-thirds of the maintenance work is done inside the ISS compared to outside the ISS. Additionally, the presented statistics indicate that, outside the ISS, robots perform about twice the number of operations in a given time period compared to astronauts. External maintenance may be performed using the Space Station Remote Manipulator System (SSRMS)

alone or in combination with EVA [44]. The preferred approach for performing external maintenance is to use the SSRMS alone as this reduces crewmember exposure to space environment hazards. The SSRMS can be used to transfer work to the crew on board or vice versa when used in combination with EVA [44].

Table 4-1 Strategy process of maintenance for the ISS

Strategy Procedure	Maintenance Situations:
Modes:	- Intravehicular Activity (IVA) - Extravehicular Activity (EVA) - Extravehicular Robotics (EVR)
Types:	- Preventative - Corrective - In-Situ - Contingency
Levels:	- Organizational (performed on orbit) - Intermediate (performed IVA on orbit or on ground) - Depot (typically performed on the ground)
Imagery & fit checks:	- Interior and exterior component photos and videos - Measure the fit of the hand tools on the hardware when building an element on the earth.

4.1.2. Robotic application in space

NASA and the Canadian Research Council (NRC) developed the Canadarm Shuttle Remote Manipulator System (SRMS) [45], [46] for the ISS. A six-twelve degree-of-freedom (DOF) programmable robotic manipulator was created to aid astronauts on the ISS during the station's early stages of development [45]. It was a tremendous success, paving the way for the employment of robotic manipulators in space applications. The many interplanetary rovers that have been dispatched to locations such as Mars are instances of cutting-edge technology that have aided in the unraveling of some of life's biggest mysteries [45]. For example, Mars rovers, such as the Curiosity and Opportunity [47], [48], employ various, customizable robotic manipulators to conduct various scientific investigations. Figure 4-1 shows four of these robotic systems, including Astrobee, Robonaut, Canadian Arm, and Special Purpose Dexterous Manipulator (SPDM).

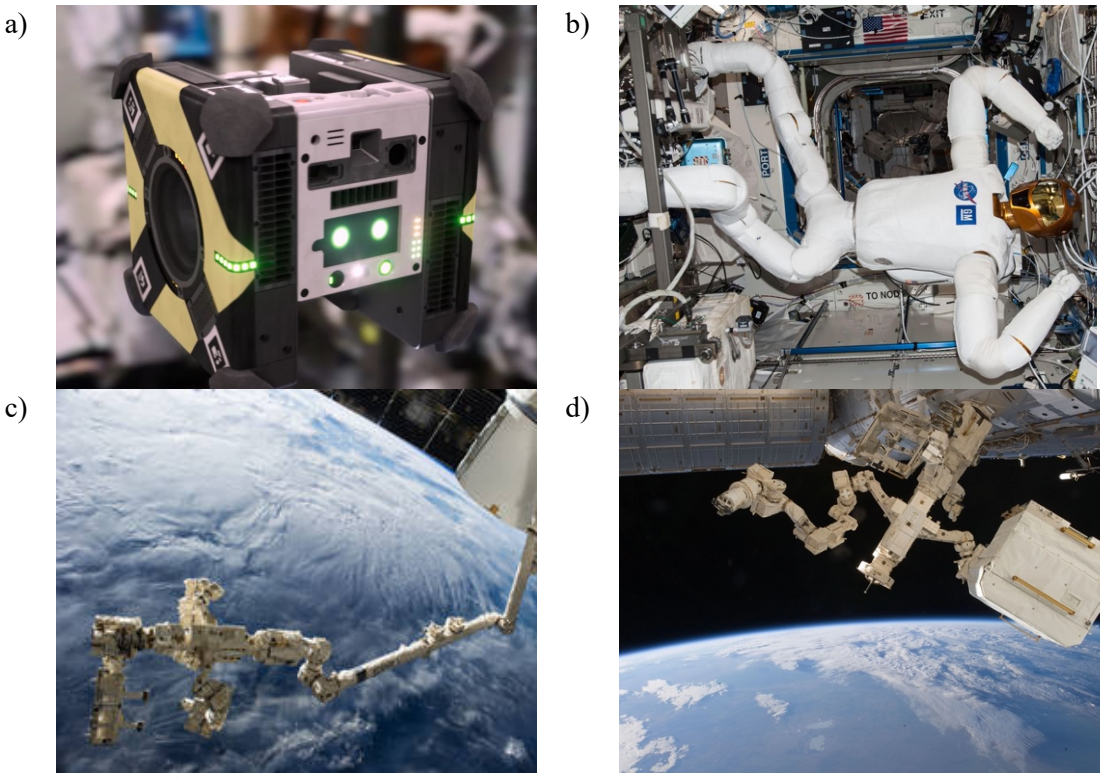


Figure 4-1 Examples of robotic applications in space. a) Astrobee [49],b) Robonaut [50], c) Canadian Arm with Dextre [51], and d) Dextre which is known as SPDM [52]

Despite the efforts that have been made to automate extravehicular activities in the space station, not many efforts have been made to automate IVA to the best of our knowledge. Developing an intelligent visual control system to detect and grasp objects with a robotic arm inside the space station comes with a significant challenge: the objects are not guaranteed to remain fixed relative to the space station body. This dynamic motion behavior of the object can occur for several reasons such as presence of zero or micro gravity and other perturbations. This and other factors necessitate the need for a sensor-directed robotic task planning system. Figure 4-2 depicts the functional architecture of such a system for space applications. This thesis focuses on developing a simulation of such a system that can be used to develop motion and task planning for robotic interaction with a 3D printer for space applications. The world modeling part of the simulation would use a model of the US lab and a robotic system interacting with a custom payload-sized 3D printer. Although geometric model of other space habitats can also be easily developed and replace those of the US lab and still utilize the robotic task planning approach developed here.

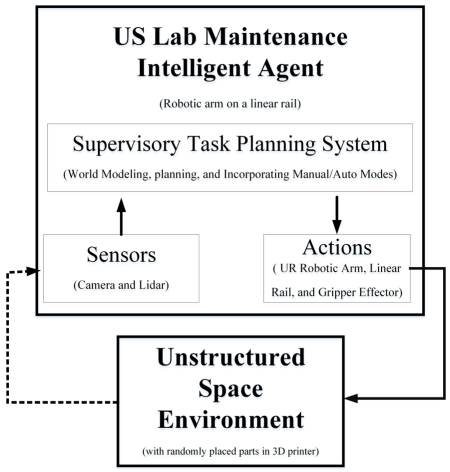


Figure 4-2 Functional block diagram of an intelligent robotic task planning system for space applications

The task planning method presented here is intended to improve the task planning capabilities of robotic systems in the unstructured environment of space habitats, and the simulation modeling is limited to a US lab and interaction with a custom payload-sized 3D printer. The simulation system developed here models a depth camera in an eye-in-hand configuration to provide immediate feedback for the controller and to help map the world for the robot's task planning. The camera sensors are used to fine-tune the motion plan during interactions with the objects to be handled and to address changes in object configurations due to perturbations that can occur in the robot task space. As a result, this sensor-directed robotic system with task planning capabilities can be used for simulation and verification of several OBM and maintenance operations.

The functional architecture of a sensor-directed task planning system for a simulated space manufacturing application is depicted in Figure 4-3. In such a system, the sensors provide information for the perception and task controller. The planner manages the sequence of actions necessary to complete a given task objective (e.g., pick up an object). The planner collaborates with the task controller, which receives robot sensor data from the robot perception module for object detection and location purposes. Once the task is ready to be executed, the task controller sends movement requests to the robotic execution module which manages motion planning and execution of robot movements.

The system uses two types of sensors: robotic sensors and world sensors. Robotic sensors are attached to the robots, and world sensors are stationary in the environment. Both sensor types can be used to update the world model, which is necessary for both task and motion planning purposes. Our simulation made use of world sensors for monitoring purposes. The robot perception module uses a deep learning algorithm to process camera sensor data in order to detect and identify objects in view, and to calculate their positions relative to the robot.

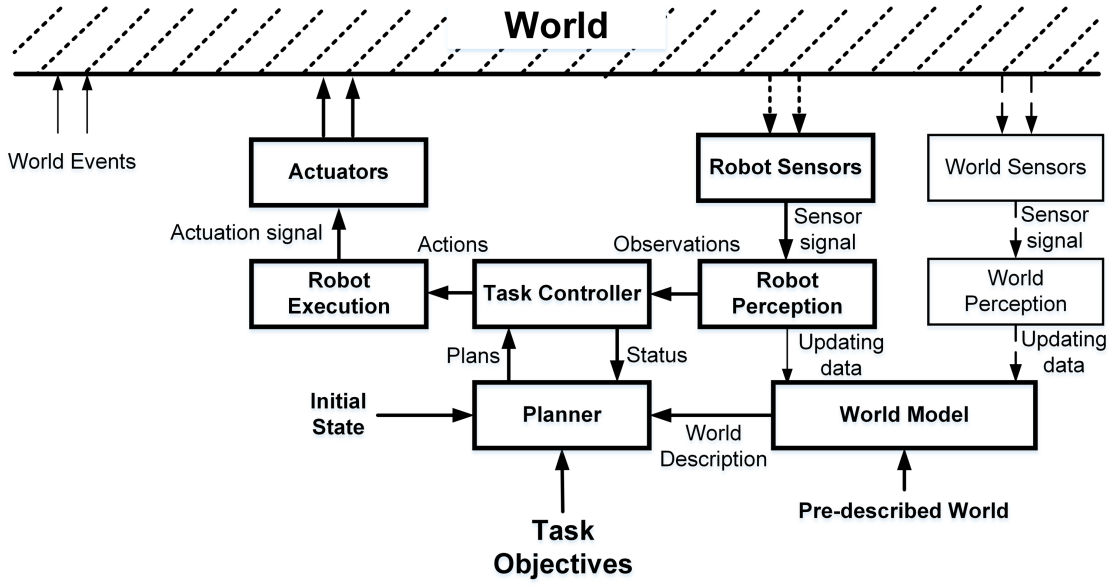


Figure 4-3 Sensor-directed task planning system for a simulated space manufacturing application

4.2. Digital Modeling for Physics Based Simulation

This section discusses the overall digital modeling framework for the physics-based robotic simulation system developed for robotic task planning and verification. The modeling system developed provides a digital representation of the robotic environment and incorporates sensory data from simulated cameras. As an example for demonstration purposes, the modeling and simulation system will be used to simulate

robotic interactions with an OBM 3D printing in a physics-based digital simulation. The simulation model serves two purposes: to demonstrate the usability of a simulation environment for task verification, and to demonstrate the use of deep-learning object detection in robot task planning developed in this thesis. The robot environment considered includes a robot on a linear rail, a perception system, a 3D printer, a 3D printed part, a tool rack, and payloads.

Figure 4-4 depicts the simulation environment which includes the US Lab as an example of a space habitat with all the robotic components. The robot manipulator is a Universal Robots UR5, a six DOF universal robot. The manipulator's base is attached to a linear rail. The gripper is mounted on the end of the robot manipulator. The linear rail is attached to the ceiling of the space station and provides a full range of linear motion for the robot arm to travel and grab objects. The gripper and the linear rail each provide one DOF which makes our whole robotic system to have eight DOF. The perceptual system provides critical information for the system task planner. The perceptual system is an RGBD (Red-Green-Blue-Depth) camera that is attached to the top of the robot's end effector. Through this camera the state of the space station can be updated frequently to provide feedback for space monitoring and robot motion planning.

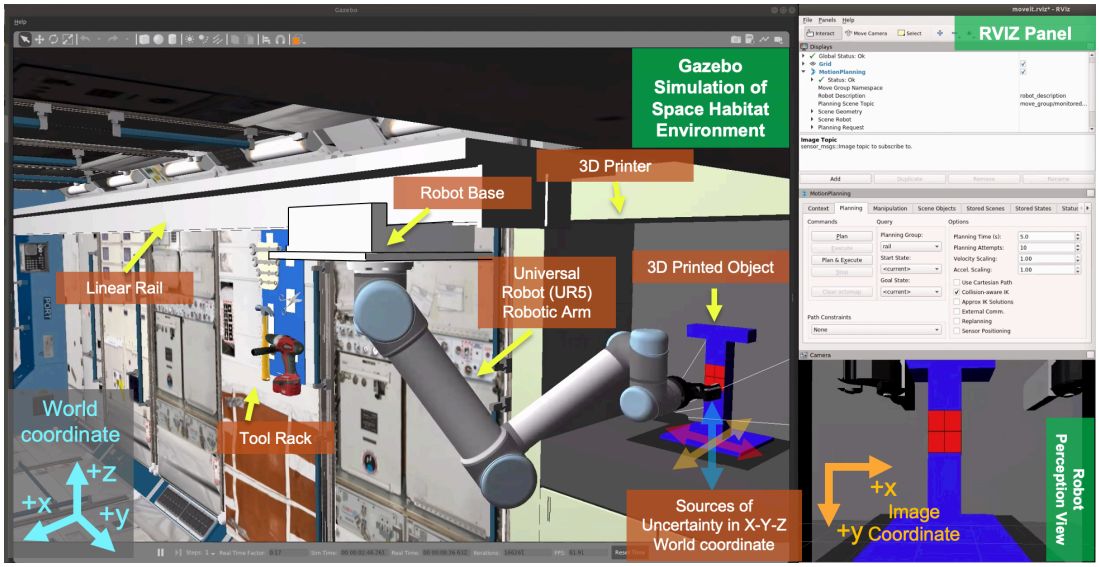


Figure 4-4 Simulation environment of the US Lab in ISS equipped robotic systems

The overall simulation-based task planning system is implemented on the free, open-source ROS platform [27]. The physics engine used for physical simulation is discussed next followed by the ROS platform implementation details.

4.2.1. Physics engine

The physics engine used in our work is Gazebo which is a 3D robotic simulator capable of simulating populations of robots in complicated indoor and outdoor contexts [53], [54]. It is comparable to other physical simulation engines in that it provides high quality simulation and supports a collection of sensing devices and interfaces for users and programs. Gazebo is comprised of five core elements: a world file, models, Gazebo Server, Graphical Client, and Gzweb. Our simulation's world file contains all the simulation's parts, such as robots, lighting, sensing devices, and stationary objects. As an example, we are modeling a robot on a linear rail system so that it can have base mobility to reach more areas within its work environment. Since the base movement on the rail is spate from the joint motion control for the robot this would make the simulation system more versatile in that it would need to handle device parallelism. Our simulation uses URDF (Universal Robot Description Format) models for the robotic manipulator and the gripper, and an SRDF (Semantic Robot Description Format) model for the linear rail. Gazebo Server is the core physics engine which simulates the environment specified by the world file. The Graphical Client communicates with the Gazebo Server to render the simulation for the user. Lastly Gzweb is a web-based client that makes use of WebGL.

4.2.2. Subsystem modeling

Our digital model for the space station environment, like many others in a robotic system, is split into numerous independent subsystems. These subsystems can be used as a tool for task planners to maintain the space station and increase its self-sufficiency, through the use of planning, computer vision, and

grasping operations. Each of these subsystems is composed of one or more modules which, in turn, contain various ROS packages and/or ROS nodes. The use of space station as the robot world is only arbitrary and is chosen as an example here. Models of any other future space habitats can easily be used instead, and the basic ingredients of the simulation-based task planning system would remain the same.

One of the benefits of using ROS is its strong support for communications between components in a robotic system. ROS communications is organized around a message-passing model in which messages are passed between ROS "nodes". Each of the modules in our digital model contain various nodes to support inter-module communications.

Figure 4-5 shows three of the modules that are used for the system task planning. The camera module includes the camera driver, image processing, position estimator, and object tracking. The motion plan module handles the coordinate transformation, inverse kinematics, path correction, and joint space. The robot module holds the control loop, state publisher, drivers and GUI (Graphic User Interface). These components are contained within the modules. For instance, the image processing functionality may be used by the motion plan module for task execution or as a controller feedback.

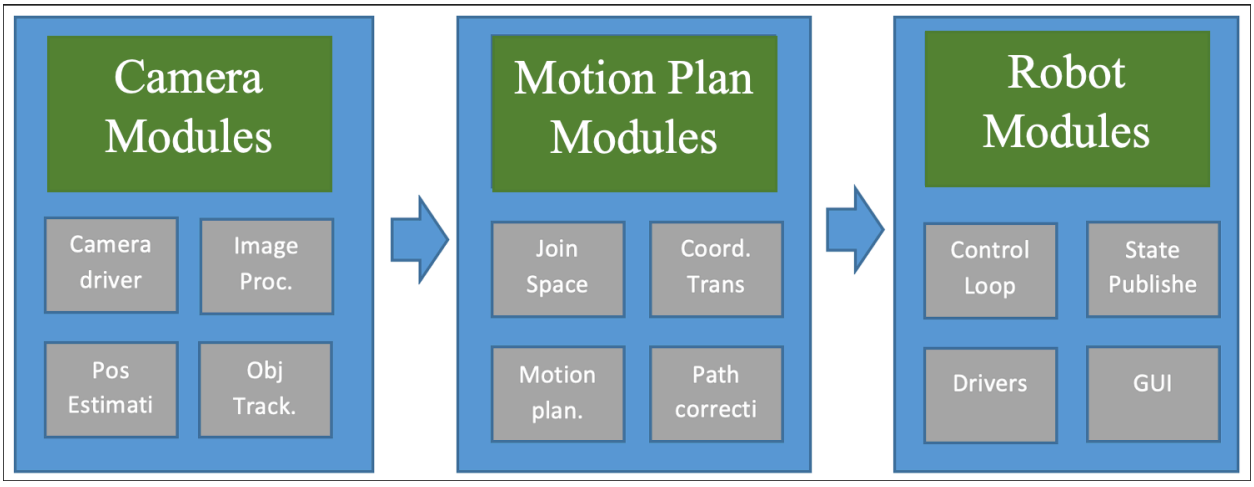


Figure 4-5 Intelligent packages used the modeling of the space autonomy world

4.2.3. ROS Nodes

In a ROS system, a node is an independent entity that executes a computation or function and communicates with other ROS nodes [55]. The graphical abstract representation of all these nodes and edges (i.e., subsystems and their connections) is referred to as a *graph* [55]. Although things can become complicated, ROS nodes are generally Portable Operating System Interface (POSIX) processes, and edges are Transmission Control Protocol (TCP) connections that provide extra fault tolerance [55]. Figure 4-6 depicts code line examples of the nodes that are running in the simulation developed here. The first four lines represent the action server nodes for the Gripper, Manipulator, Rail, and The Image Analyzer. The rest of nodes are the Gazebo nodes, RViz nodes, state publisher, move group, and so on.

```
1.  $ rosnode list
2.  /GripperActionServer
3.  /ManipulatorActionServer
4.  /RailActionServer
5.  /ImageAnalyzerActionServer
6.  /gazebo
7.  /gazebo_gui
8.  /joint_state_publisher
9.  /move_group
10. /move_group_commander_wrappers_1634437917262890235
11. /move_group_commander_wrappers_1634437917305371204
12. /move_group_commander_wrappers_1634437917308722773
13. /robot_controllers
14. /rosout
15. /rostopic_23701_1634437929192
16. /rviz_ros_23293_2053448388210896119
17. /state_publisher
18. /testimageclient_py
```

Figure 4-6 The nodes that are running

Figure 4-7 is a partial screenshot of rqt – the ROS node graph visualization tool – which represents how these nodes are connected. For instance, the Gazebo simulator node is connected to several other nodes since the simulation is dynamic and its physical state is maintained within Gazebo.

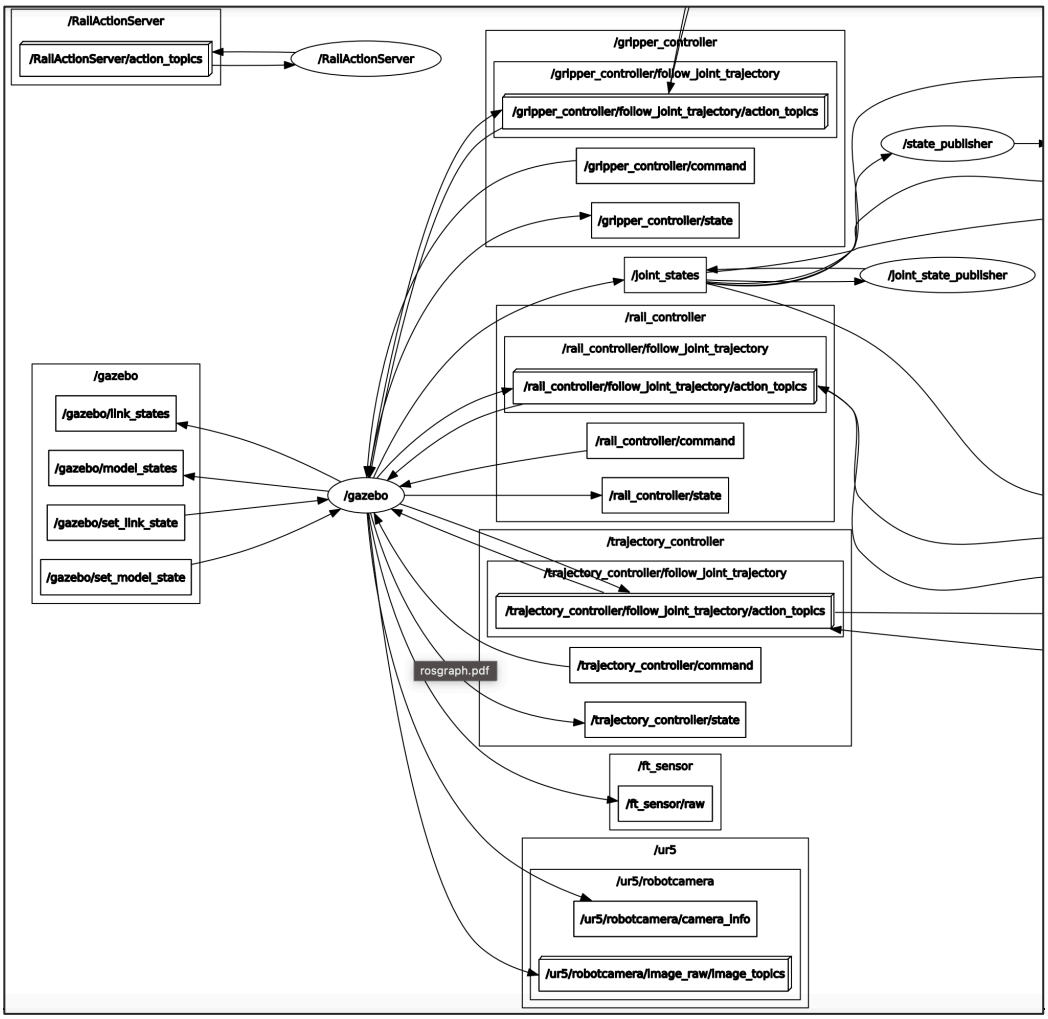


Figure 4-7 A partial screenshot of the ROS rqt graph, visualized nodes, and edges

4.2.4. Agent-based AI task planning system

The goal of an agent-based AI task planning system is to enable reliable communications between different simulation agents, as well as to process signals for robot control or monitoring purposes. Figure 4-8 shows our agent-based AI system architecture for the simulation of the space habitat environment in ROS. ROS includes low-level features such as a build system, message forwarding, device drivers, and some integrated capabilities such as navigation [55]. The core module of our Agent-based AI system is the Task Planning action client module. This module allows the user or the automated system to choose a task to execute. Each task corresponds to a series of goals for the robot components. The Task Planning action client module

communicates with four action server modules: Linear Rail, Gripper, Manipulator, and Image Analyzer. The Linear Rail, Gripper, and Manipulator action server modules interface with the MoveIt library. MoveIt is the main source of manipulation (including mobile manipulation) functionality in ROS for processing and executing planning requests [56]. Lastly, the Image Analyzer module interfaces with the OpenCV library. OpenCV is a real-time optimized Computer Vision library that provides various image processing functions [57].

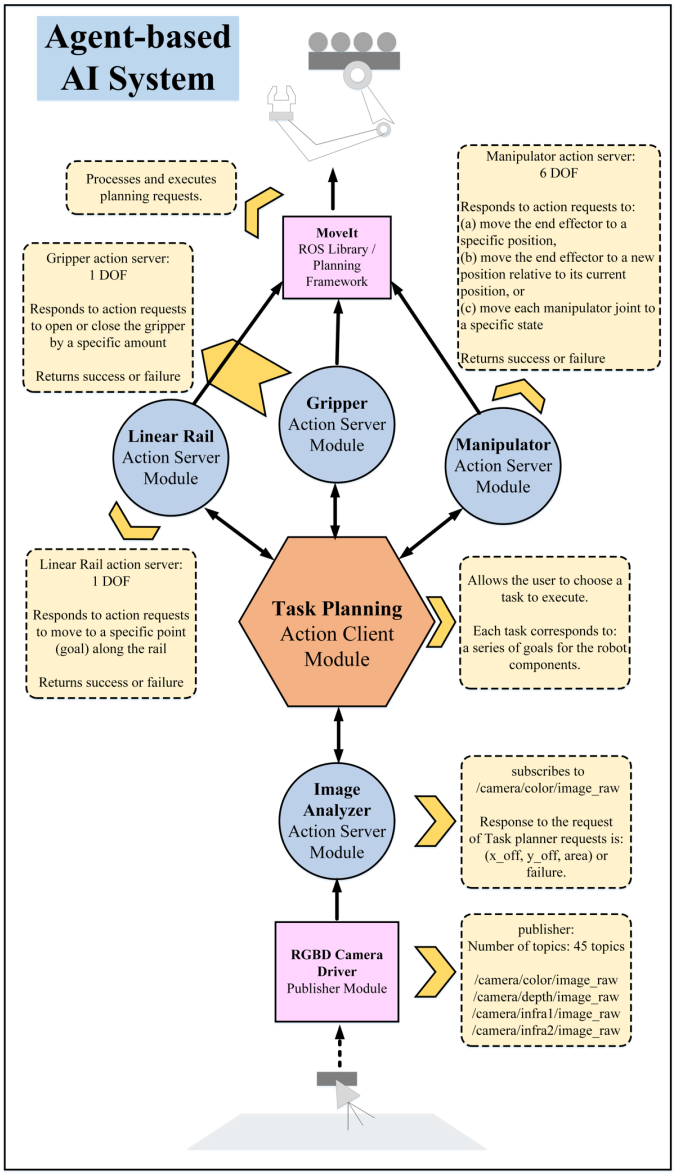


Figure 4-8 Agent-based AI system architecture for the simulation of the space habitat environment

4.2.5. ROS Actions

Our system's task planner is based on a set of primitive actions for task planning verification. These actions are implemented as ROS actions.

In ROS, actions are a commonly used client-server communications paradigm for long-running tasks, and are composed of three components: a goal, feedback, and a result [55]. The goal component can be thought of as a command sent by an action client to instruct the action server to execute a particular task, the feedback component is a periodic communication from the action server to inform the action client of the status of the task being executed, and the result component is returned by the action server to the action client to communicate the result of the task upon its completion. These underlying communications use the ROS Action Protocol which is based on the lower-level node-based ROS communications functionality. Actions in ROS are similar to services, but actions can be preempted (i.e., the user can abort the execution), and they additionally can provide consistent, timely feedback as the execution progresses, in contrast to services that only deliver a single response at the end of execution.

4.2.6. Action Server Modules

Table 4-2 tabulates all the action server nodes and messages. The Manipulator action server module commands a 6-DOF UR5 robotic manipulator. The module responds to action requests to (a) move the end effector to a specific position, (b) move the end effector to a new position relative to its current position, or (c) move each manipulator joint to a specific state. The Gripper action server module commands a 1-DOF two-finger robotic gripper. The module responds to the action requests to open or close the gripper by a specified amount. The Linear Rail action server module commands a 1-DOF linear rail. The module responds to action requests to move to a specified point (goal) along the rail and returns success or failure. All three of the movement-based action server modules return a status of success or failure upon completion of movement.

The Image Analyzer action server module implements the perception system. It subscribes to image topics from the camera driver and responds to the Task Planning action client module to analyze the current camera image. The image analyzer model returns the offset, in pixels, of the target's center from the center of the camera image and the area, in square pixels, of the target's bounding box.

Table 4-2 Action server nodes and messages

ACTION SERVER	Message Definition	Goal Request Task	Robot Element	Action Type	ROS Message

Manipulator	`Manipulator.action:`
	`#goal action`
	`string goal type`
	`float32 p0`
	`float32 p1`
	`float32 p2`
	`float32 p3`
	`float32 p4`
	`float32 p5`
	`float32 p6`
	`---`
	`#result`
	`uint32 val`
	`---`
	`#feedback`
	`uint32 status`
		Move the end effector to a specific position
		Return: success or failure
			End effector	Position (m) Orientation (rad) Goal	`

		Move the end effector to a new position relative to its current position Return: success or failure	End-effector	Relative Position(m) Goal	`'pose_rel'` `position offsets` (P_0, P_1, P_2): (dx, dy, dz)
		Move each manipulator joint to a specific state Return: success or failure	Manipulator Joints	Joint Angle(rad) Goal	`'joint'` `joint angles` $(P_0, P_1, P_2, P_3, P_4, P_5)$: $(\theta_1, \theta_2, , \theta_3, \theta_4, \theta_5, \theta_6)$

Gripper	`Gripper.action:`
	`#goal action`
	`float32 p0`
	`---`
	`#result`
	`uint32 val`
	`---`
	`#feedback`
	`uint32 status`
		Open or close the gripper by a specific amount
		Return: success or failure
			Gripper Fingers	Position(m) Goal	`'val'`
					`- local position` (P_0): (x)

Rail	`Rail.action:`
	`#goal action`
	`float32 p0`
	`---`
	`#result`
	`uint32 val`
	`---`
	`#feedback`
	`uint32 status`
		Move to a specific point(goal) along the rail
		Return: success or failure
			Robot Base	Position(m) Goal	`'dist'`
					`- position` (P_0): (x)

Image Analyzer	`IA.action:`
	`#control action`
	`int32 control_type`
	`---`
	`#result`
	`float32[] result`
	`---`
	`#feedback`
	`uint32 status`
		Analyze the current camera image
		Return: offsets [pixels] and area [square pixels]
			Robot Perception	Image Analysis	`'control-type'`
					`- trigger`
					`val: '1'`

Figure 4-9 depicts an example code of an action client sending a joint goal to the Manipulator action server. The code first specifies a goal type (joint) and six joint angles, then sends the goal to the action server for execution.

```
goal =
msg.ManipulatorGoal(
    goal type='joint',
    p0=rad0,
    p1=rad1,
    p2=rad2,
    p3=rad3,
    p4=rad4,
    p5=rad5
)
client.send_goal(goal)
```

Figure 4-9 Action client: sending a joint goal

The RGBD camera driver is a publisher module. The camera sensor captures data from the environment and publishes 45 topics, including the four listed in Figure 4-10. These four topics provide streams of data from their respective camera sensors (color image, depth, infrared images).

```
/camera/color/image_raw
/camera/depth/image_raw
/camera/infra1/image_raw
/camera/infra2/image_raw
```

Figure 4-10 Published topics by the RGBD camera driver

Figure 4-11 depicts three running action servers in the task planning system: gripper, manipulator, and rail. Each action server module communicates with its respective client through ROS topics. Action clients use the “goal” and “cancel” topics to request execution of a goal, and to cancel a running execution. Action servers use the “status”, “result”, and “feedback” topics to communicate goal status and execution results.

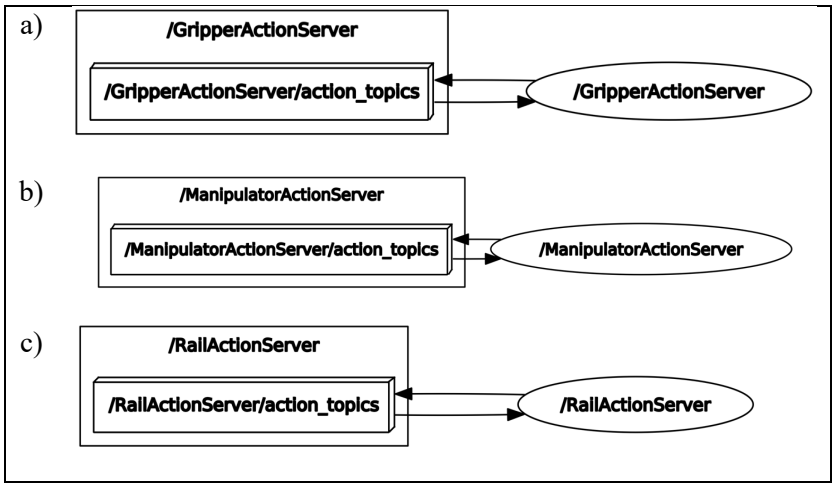


Figure 4-11 Robot action servers for the 3D printer interaction: a) Gripper b) Manipulator c) Rail

4.2.7. Primitive actions

Primitive actions facilitate task-based robot programming. Primitive actions are programmed only once and allow the user to utilize them multiple times. For instance, closing and opening the gripper is a type of primitive action. Another example of a primitive action is moving the robot end effector by increments (e.g., right, left, up, down, forward, or backward).

4.2.8. Higher-level actions

Higher-level action commands combine several primitive actions. For instance, several primitive actions are involved in opening the door of a 3D printer:

1) Adjust the gripper to an appropriate value to capture the door handle.
2) Move the rail to near where the door handle is when the door is closed.
3) Move the manipulator to the “grasp” position for the door handle.
4) Close the gripper to a value appropriate for grasp the handle.
5) Move the rail to near where the door handle is when the door is open.
6) Open the gripper to an appropriate value to release the door handle.
7) Move the manipulator back away from the handle.

4.2.9. Hierarchical task network

Figure 4-12 shows an example hierarchical task network for a robot and 3D printing interactions. On the top of the task network, the intelligent system receives the most abstract action. This action is composed of two sub-actions. These sub-actions are less abstract than the parent action, and more abstract than the actions which compose them. In the case of this particular task (3D-print a part and place it in a designated location), the top-level action contains two sub-actions: (a) bring the printed part and the plate to the desk, and (b) separate the part from the plate. Each of these tasks utilizes primitive or low-level actions as well as higher-level actions. Composing actions in this way allows the planner to call only the top-level action and reduce redundancy which is a major factor in autonomy.

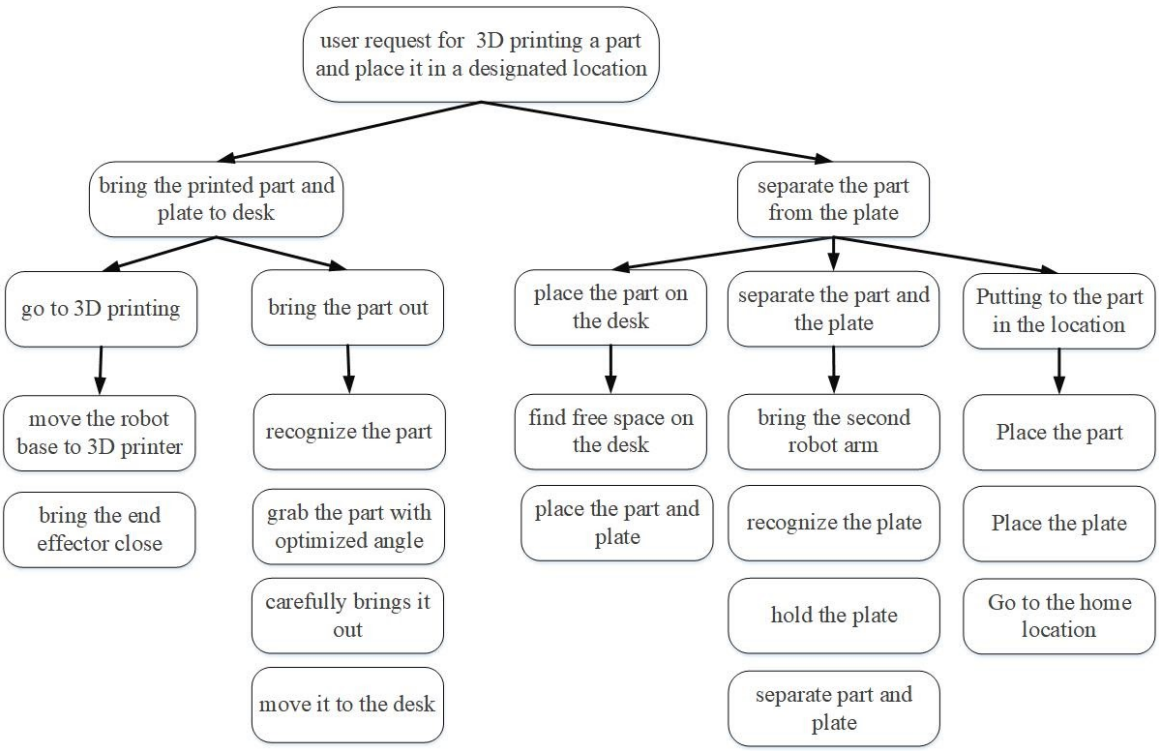


Figure 4-12 An example hierarchy structure of planning for interaction with the 3D printer

4.3. Robot Perception and Control

Using a set of predefined tasks to accomplish higher-level tasks works well in static scenarios but is insufficient in dynamic environments. For example, we may need to pick up an object that is not guaranteed to be in a particular location (e.g., floating objects in space). To handle such cases, we equipped the task planner with a perception system. The role of robot perception in our task planning system is to provide immediate feedback for the task executor. Another role of the perception system can be to model and update the environment through the use of sensor data.

In this example, the perception system is involved in the control of the system when the robot arm is approaching the object. The controller utilizes the data steaming from the camera and tracks the object until the object is centered in the gripper. Then it estimates the depth through analysis of the 2D RGB image data. The system also monitors the depth information from the camera sensor. We explicitly do not use this feature for robot control, but it can be beneficial in further advancement of the perceptual system. According to [58], in comparison to more traditional approaches such as capacitive sensors, using depth cameras to detect touch has the following benefits:

- No instrumentation is required on the interactive surface.
- The engaging surface does not have to be perfectly flat.
- Information regarding the shape of users and their arms and hands above the surfaces may be necessary for assessing hover state or determining that numerous touches originate from the same hand or user.

4.3.1. Image Analyzer Module

The Image Analyzer action server is one of the key components of our agent-based AI system architecture. The role of the image analyzer is to monitor (via ROS subscription) the camera frames and return world

coordinate information for the robot task planning system. Our Image Analyzer utilizes a deep learning model which is called the YOLO (You Only Look Once) method to detect and locate the object. Then, it applies several interpolation routines to the image data to obtain the object's world coordinate information. A deep-learning-based approach was chosen due to its successful application in many similar problems, as well as for its extensibility to additional object types.

Figure 4-13 depicts the steps used by the Image Analyzer module, from analyzing the camera image most recently received via ROS subscription, to the final calculation of offsets and responding to the planner request. Upon each image analysis request from the planner, these following steps are used:

1) Preprocess the image
2) Apply YOLO deep learning detection system
3) Classify each detection by using its highest-score class confidence value
4) Filter out detections with class confidence levels below the configured threshold value
5) Apply NMS (Non-Max Suppression) filter
6) Filter out detections with undesired class types
7) Calculate the detection bounding box corners and area
8) Determine the detection with the largest bounding box area
9) Calculate the bounding box offset vector from image center
10) Respond to the planner request with the offset vector and area

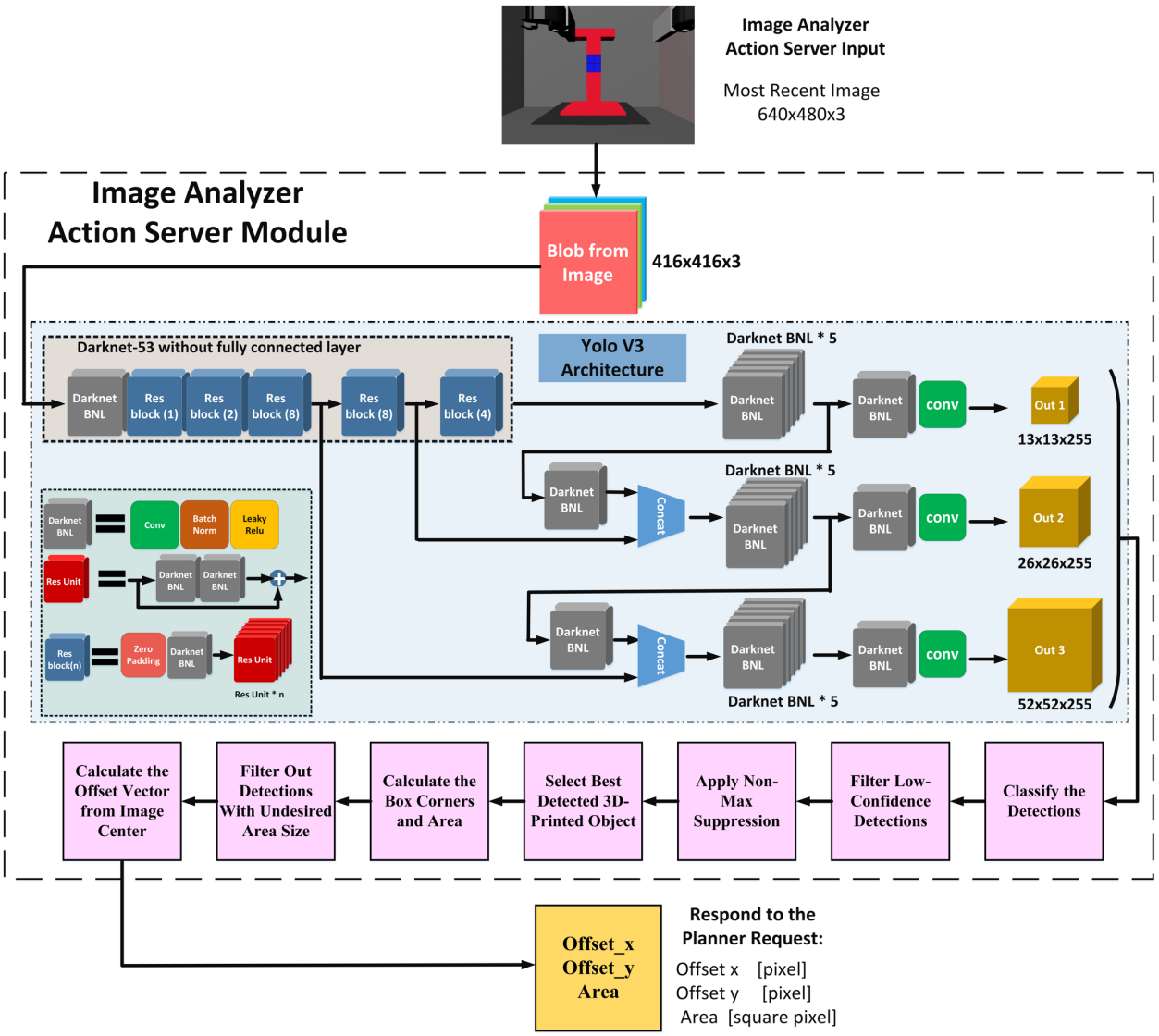


Figure 4-13 Image Analyzer action server module

4.3.2. Image preprocessing

The input image is the most recent image from the camera driver that captures frames from the real time simulation. The input size is $(640 \times 480 \times 3)$. First, the image is transformed into a "blob" using the OpenCV call blobFromImage [59]. In our case, a "blob" is a binary object that represents an NCHW matrix based on the input image. An NCHW matrix is a data format that represents N images, each with C channels,

a height of H pixels, and a width of W pixels. In our case, N is 1, C is 3, and H and W are each 416 (blobFromImage scales the image non-proportionally).

4.3.3. Deep learning model for object detection

After image preprocessing, the second stage is the application of a deep learning detection system to find the object and identify its bounding box. The detection system used here is YOLOv3 [60]. The YOLOv3 feature extractor, Darknet-53 (it has 53 convolutions), is inspired by ResNet and FPN (Feature-Pyramid Network) architectures. As can be seen in Figure 4-14, Darknet-53 contains skip connections (like ResNet) and three prediction heads (like FPN), each processing the image at a different spatial compression. This architecture is flexible, allowing for the detection of objects with different sizes. Each of the network outputs produces a matrix. Together, these matrices represent all the detected objects.

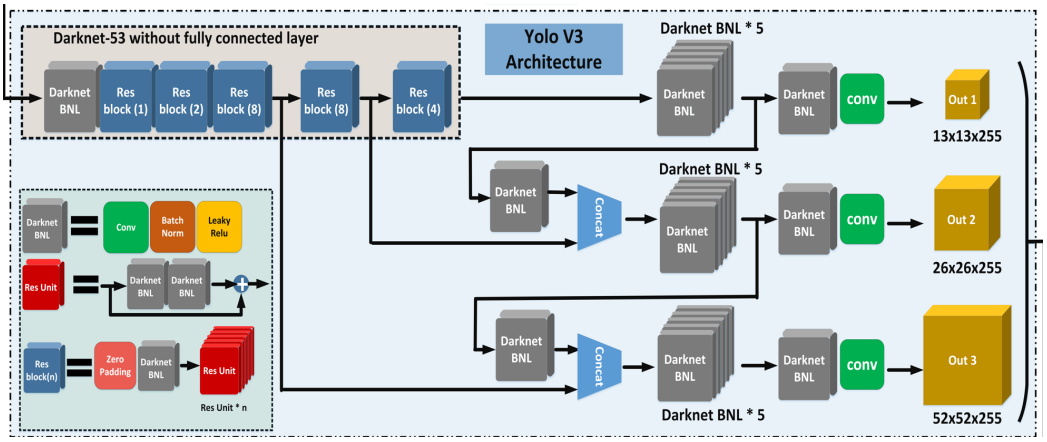


Figure 4-14 YOLOv3 architecture

For an input image geometry of 416x416x3, the sizes of these matrices are $(13 \times 13 \times 255)$, $(26 \times 26 \times 255)$, and $(52 \times 52 \times 255)$. For our application, the multi-phase architecture of YOLOv3 provides a great advantage because the object's size may vary considerably based on its distance from the camera. The output in each of these cells is as follows:

```
[x, y, w, h, obj confidence, score label 1, score label 2, …, score label n]
```

The first four cells in each row are the x, y, width, and height of the bounding box, and the fifth cell is the bounding box confidence. The rest of the cells—from the fifth cell through the end—are the confidence scores for each of the classes.

4.3.4. Post processing of object detection

From the output of the second stage, we classify each detection by using its highest-score class confidence value. Then, a confidence threshold filter is applied to eliminate low-confidence detections (i.e., below 0.5). The next (fifth) stage applies Non-Max Suppression to "suppress" all non-optimal bounding boxes and preserve the optimal bounding box which contains the object. For this, we chose an NMS threshold of 0.3 and an NMS box score threshold of 0.5. After the NMS stage, any detections whose class types are not that of the 3D-printed object are discarded. The bounding box corners and area are then calculated, and only the detection with the largest bounding box area is retained. Finally, the offset vector from the image center to the bounding box center is calculated and returned, along with the area value, to the planner.

4.3.5. Data Collection and Training

To generate the dataset used to train our model, images were gathered using the Gazebo simulation. Three hundred and seventy-seven images from the robot arm camera were captured. Some examples of these captured images are depicted in Figure 4-15. We collected images that met one of the three conditions: (a) images which do not include the 3D-printed object, (b) images which are edge cases (i.e., in which the object is partially obscured), and (c) images which include a full, unobscured view of the object.

a)

b)

c)

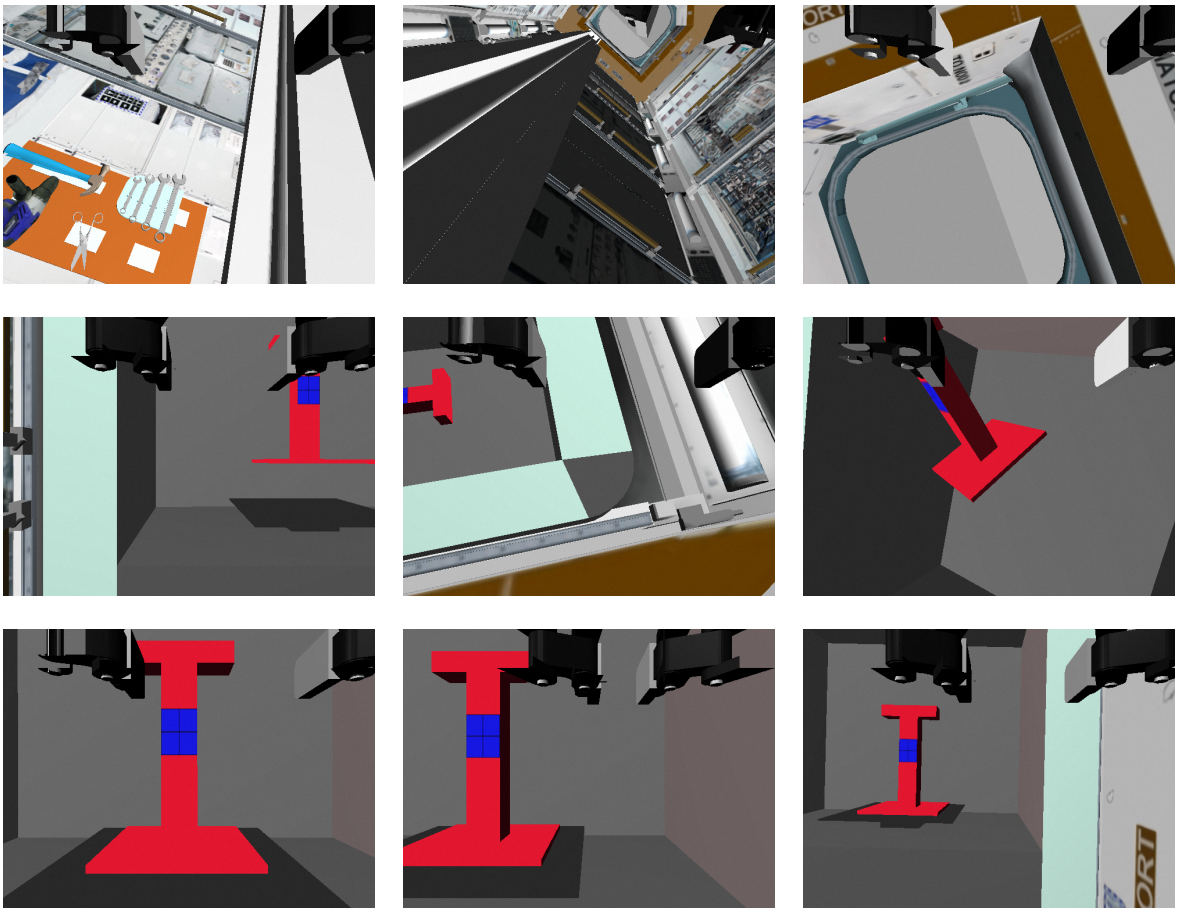


Figure 4-15 Collected data within the Gazebo simulation a) No 3D-printed object, b) Edge cases -partially visible object c) Unobscured 3D-printed object

These images were split randomly into two groups of training and testing with respective split ratio of 70% and 30%. Although this split generation was distributing randomly, it was ensured that each of these image conditions was fairly represented in both the training and the testing groups.

The training detection objects are:

- `obj:` The (red) 3D-printed object
- `targ:` The object's (blue) gripper section
- `lgrip:` Left gripper finger
- `rgrip:` Right gripper finger

Training was done in a YOLO training environment using a transfer learning technique. Pre-trained weights for 73 convolutional layers were transferred to the model for extra training. Table 4-3 summarizes the training parameters that were used.

Table 4-3 Summary of training parameters

Testing	Training	Input Image	Model Parameters	Training Parameters:
batch = 64 subdivisions = 16	batch = 64 subdivisions = 16	width = 416 height = 416 channels = 3	momentum = 0.9 decay = 0.0005 angle = 0 saturation = 1.5 exposure = 1.5 hue = 0.1	learning_rate = 0.001 burn_in = 1000 max_batches = 8000 policy = steps steps = 6400,7200 scales = 0.1,0.1

Figure 4-16 illustrates the results of training a YOLO model which is trained for 8000 epochs. We chose to train our model for 2400 epochs, as this resulted in optimal mAP and loss values for all four objects.

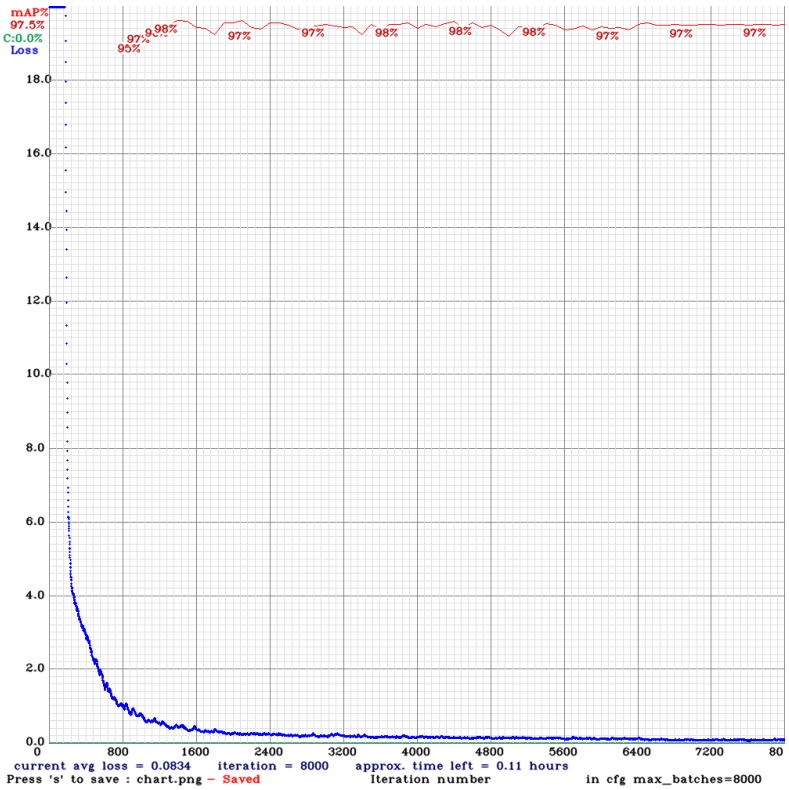


Figure 4-16 Model training results

The selected trained model has a mAP of 96.17% for the 3D-printed object, 99.23% for the target, 96% for the left gripper finger, and 98.31% for the right gripper finger. Table 4-4 lists the training results (mAP, TP, and FP) for each object class.

Table 4-4 Training results for four object classes

Class ID	Tag Name	mAP	TP	FP
0	obj	96.17%	75	1
1	targ	99.23%	67	2
2	lgrip	96.33%	120	3
3	rgrip	98.31%	118	3

4.3.6. Robot Perception System

The primary purpose of our robot's visual perception system is to identify and locate the 3D-printed object and, via image analysis, to estimate its position so that the manipulator can accurately approach and grasp the object. To accomplish this, the current camera image is analyzed to calculate an offset vector that represents the 2-dimensional offset (in the camera image plane) from the image center to the center of the 3D-printed object's bounding box (see Figure 4-17). In addition, the area of the 3D-printed object's bounding box is calculated. The bounding box area is used to estimate the distance (in meters) from the camera to the object, and the offset vector (along with the estimated distance value) is used to estimate the object's position in the camera frustum. This object position value is then used to calculate the position goal (in world space) for the robot arm's end effector.

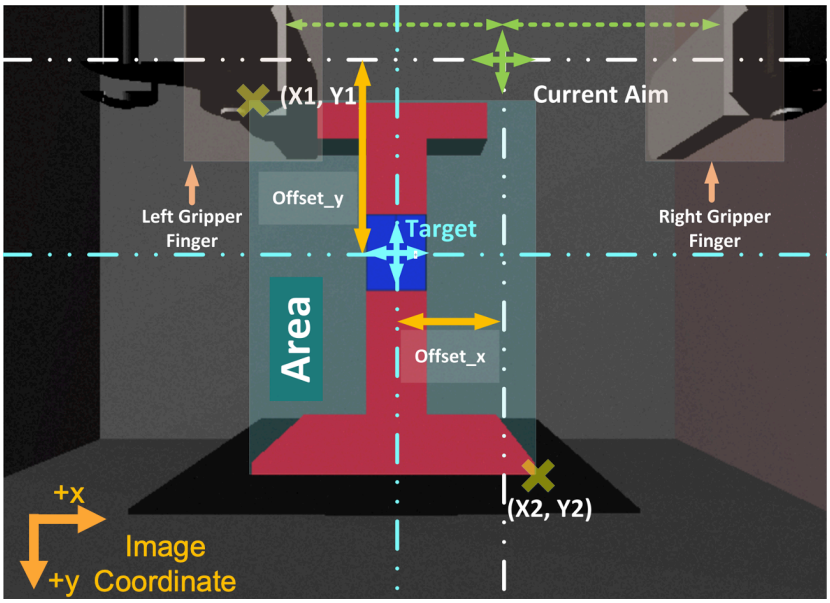


Figure 4-17 Robot perception view of the inside of the 3D printer

4.3.7. End Effector Positioning

For the positioning of the end effector (for 3D-printed object approach), it is assumed that both the object and the camera are aligned with the world coordinate system. In this case, the task is reduced to using the bounding box area and the X and Y coordinates (in image space) to calculate the X, Y, and Z of the object in world space. Figure 4-18 illustrates the relationship between image-space X and Y (pixels) and world-space Y and Z (meters).

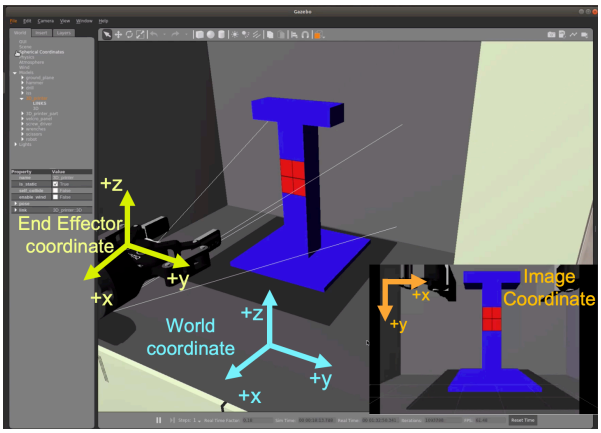


Figure 4-18 Coordinate systems shown on the simulation snapshot including world, robot, and image coordinates

4.3.8. Estimation of X Distance

Our Python linear interpolation function is listed in Figure 4-19. To use the bounding box area to estimate the distance (along X) from the 3D-printed object to the camera, we linearly interpolate along a piecewise linear curve (Figure 4-20) constructed from a set of known bounding box area → distance mappings) collected from the Gazebo simulation.

```
1.  def linterp(xlist, ylist, xval):
2.      xval = float(xval)
3.      num = len(xlist)
4.      if (num != len(ylist)) or (num < 2):
5.          raise ValueError('invalid list length(s)')
6.      xsort = xlist[:]
7.      xsort.sort()
8.      if (xlist != xsort):
9.          raise ValueError('xlist not sorted')
10.     xset = set(xlist)
11.     if (num != len(xset)):
12.         raise ValueError('xlist contains dupe(s)')
13.     for i in range(0, num-1):
14.         xa = float(xlist[i])
15.         xb = float(xlist[i+1])
16.         if (xval >= xa) and (xval <= xb):
17.             ya = float(ylist[i])
18.             yb = float(ylist[i+1])
19.             if (xval == xa):
20.                 return ya
21.             if (xval == xb):
22.                 return yb
23.             m = (yb - ya) / (xb - xa)
24.             return ya + (m * (xval - xa))
25.     if (xval < xlist[0]):
26.         xa = float(xlist[0])
27.         ya = float(ylist[0])
28.         xb = float(xlist[1])
29.         yb = float(ylist[1])
30.         m = (yb - ya) / (xb - xa)
31.         return ya + (m * (xval - xa))
32.     elif (xval > xlist[num-1]):
33.         xa = float(xlist[num-2])
34.         ya = float(ylist[num-2])
35.         xb = float(xlist[num-1])
36.         yb = float(ylist[num-1])
37.         m = (yb - ya) / (xb - xa)
38.         return ya + (m * (xval - xa))
39.     raise ValueError('unexpected state')
```

Figure 4-19 Linear piecewise interpolation Python code

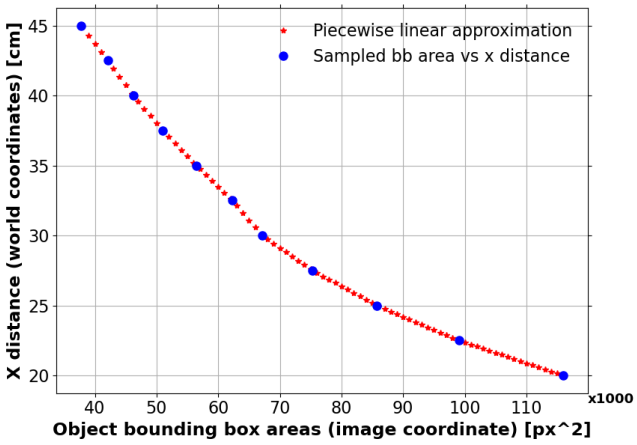


Figure 4-20 Robot perceptual system calibration from object bounding box area to X world coordinate

4.3.9. Estimation of Y and Z Distance

To use the X and Y offsets (in image space pixels) of the 3D-printed object's bounding box to estimate the Y and Z distance from the 3D-printed object to the center of the camera frustum, we first estimate the scaling factor between image space and world space for the current camera image. This scaling factor is estimated by linearly interpolating along a piecewise linear curve (see Appendix F) constructed from a set of known X distance → scaling factor mappings) collected from the Gazebo simulation.

4.3.10. Multi-step vs 1-step Approach

If the object's position is known, it is a relatively simple task to move the robot arm to the object. Our goal is to utilize the robot arm's sensory information to detect and locate the object. We have implemented our approach in a function called `auto_position_controller`. When this function is called, the robot arm's camera image is analyzed, the object's position is estimated, and the robot arm is advanced toward the object. This approach works but we have found that, due to initial camera/object alignment and object distance from camera, the estimate is not always within tolerances. To address this, a 2-step approach was developed that first uses the results of the image analysis to move the robot arm in the Y/Z plane such that

the object is centered in the camera view. Then the image analysis is performed again using this view, and the results used to advance towards the object. This 2-step approach significantly reduces the positioning error but was found to occasionally misestimate the X distance. This is believed to be caused by the fact that the object in the camera view is smaller when the object is further from the camera, causing bounding box geometry errors to have amplified effects. To address this problem, a 3-step approach was developed that adds another X-estimation step, allowing the X distance to be recalculated once the camera is closer to the object. It is expected that adding more steps (e.g., 4-step, 5-step) would further decrease error, but with diminishing returns.

The 1-step approach is as follows:

1) Perform image analysis and use the returned offset values to calculate the required Y/Z movement to center the object in the camera view; then use the bounding box area to calculate the required X movement to bring the end effector to just outside the object's gripping target.
2) Execute that X/Y/Z movement.

The 2/3-step approach is as follows:

1) Perform image analysis and use the returned offset values to calculate the required Y/Z movement to center the object in the camera view.
2) Execute that Y/Z movement.
3) (3-step only): Perform image analysis and use the returned offset values to calculate the required Y/Z movement to further center the object in the camera view, and the returned bounding box area to calculate the required X movement to bring the end effector to within a "buffer zone" (currently 0.25 m) of the object.
4) (3-step only): Execute that X/Y/Z movement.
5) Perform image analysis and use the returned offset values to calculate the required Y/Z movement to further center the object in the camera view, and the returned bounding box area

to calculate the required X movement to bring the end effector to just outside the object's gripping target.

6) Execute that X/Y/Z movement.

7) Execute a predefined "grasping" movement that embraces the object's gripping target in the gripper.

The Python code for the auto position controller is depicted in Figure 4-21.

```
1.  def auto_position_controller(ia_client, mani_client, center_grip,
2.  x_buf):
3.      offsets = call_image_analyzer(ia_client)
4.      if (offsets is None) or (len(offsets) == 0):
5.          return None
6.      (x_off_pix, y_off_pix, area) = offsets
7.      est_delta_x = linterp(X_OBJAREA, X_XREL, area)
8.      est_yz_absslope = linterp(YZ_X_REL, YZ_ABSSLOPE, est_delta_x)
9.      y_slope = est_yz_absslope * (-1.0)
10.     z_slope = est_yz_absslope
11.     if (center_grip):
12.         est_delta_y = y_slope * x_off_pix + (Y_TARG_XHAIRS -
13. Y_TARG)
14.         est_delta_z = z_slope * y_off_pix + (Z_TARG_OBJCENT -
15. Z_TARG)
16.     else:
17.         est_delta_y = y_slope * x_off_pix
18.         est_delta_z = z_slope * y_off_pix
19.     if est_delta_x is None:
20.         return None
21.     if est_delta_y is None:
22.         return None
23.     if est_delta_z is None:
24.         return None
25.     if (est_delta_x > x_buf):
26.         x_rel_goal = (est_delta_x - x_buf) * (-1.0)
27.     else:
28.         x_rel_goal = 0.0
29.     y_rel_goal = est_delta_y * (-1.0)
30.     z_rel_goal = est_delta_z * (-1.0)
31.     result = mani_pose_rel_goal(mani_client, x_rel_goal,
32. y_rel_goal, z_rel_goal)
33.     return result
```

Figure 4-21 Auto position control Python code

4.4. Simulation Task Planning and Image Analyzer Results

4.4.1. End effector positioning error Analysis

The results of the end effector positioning based on three different positioning approaches are tabulated in Table 4-5. The table columns are the test ID, end effector starting position, end effector ending position, and the error. Test ID has two subparts, one is the X distance from the 3D-printed object to the camera, and second is a mnemonic that represents the relative position of the 3D-printed object from the robot gripper. For example, "45-UR" implies a test position 45 cm (in X) from the camera, and that the object is on the upper right side of the camera view. Note that the distances from the 3D-printed object are not available to the planner. Seven test cases have been chosen from a variety of positions relative to the object.

Table 4-5 Results of the end effector positioning in the simulation environment

Approach	TEST_ID	X_START[m]	Y_START[m]	Z_START[m]	X_END[m]	Y_END[m]	Z_END[m]	ERR_YZ	ERR_X	ERR_XYZ
1-step	20-LL	-0.295303	2.553749	1.534588	-0.461248	2.54472	1.498642	0.01163832	0.034036611	0.035971396
	25-UR	-0.245311	2.488706	1.464537	-0.500341	2.552178	1.497201	0.012832132	0.005056389	0.013792414
	30-LL	-0.195266	2.748711	1.584527	-0.467798	2.559068	1.509115	0.010337755	0.027486611	0.029366357
	30-BA	-0.195341	2.548711	1.509527	-0.500142	2.545381	1.499968	0.010166912	0.004857389	0.011267667
	35-UR	-0.145212	2.450246	1.409574	-0.498146	2.558669	1.501497	0.012793027	0.002861389	0.013109122
	40-LL	-0.09539	2.818674	1.63961	-0.442562	2.574457	1.520063	0.027777565	0.052722611	0.059592506
	45-UR	-0.045186	2.388817	1.359514	-0.471654	2.5559	1.491735	0.019212986	0.023630611	0.030455617
	45-CE	-0.045231	2.576658	1.550436	-0.496681	2.546246	1.503374	0.00667354	0.001396389	0.006818067
	average							0.01392903	0.019006	0.025046643
2-step	20-LL	-0.295219	2.553701	1.534596	-0.49015	2.542742	1.496075	0.014762423	0.005134611	0.015629886
	25-UR	-0.245398	2.48869	1.464521	-0.495968	2.543545	1.498732	0.012013336	0.000683389	0.012032758
	30-LL	-0.195196	2.748713	1.584483	-0.500437	2.542285	1.501108	0.010638184	0.005152389	0.01182024
	30-BA	-0.195304	2.548595	1.509507	-0.502214	2.544616	1.501258	0.009273457	0.006929389	0.011576418
	35-UR	-0.145241	2.450238	1.409664	-0.508686	2.546648	1.500711	0.009096817	0.013401389	0.016197201
	40-LL	-0.095175	2.818618	1.63952	-0.496782	2.544596	1.504	0.006937639	0.001497389	0.007097394
	45-UR	-0.045345	2.388601	1.359596	-0.505593	2.54604	1.503709	0.00644756	0.010308389	0.012158697
	45-CE	-0.045367	2.576698	1.550545	-0.502538	2.550186	1.5022	0.007504626	0.007253389	0.010437005
	average							0.009584255	0.006295042	0.0121187
3-step	20-LL	-0.295294	2.553643	1.534552	-0.502909	2.542438	1.497766	0.01337565	0.007624389	0.015396081
	25-UR	-0.245282	2.488678	1.464594	-0.491647	2.542769	1.500302	0.011019868	0.003637611	0.011604727
	30-LL	-0.195271	2.748596	1.584458	-0.495385	2.541724	1.500487	0.01147241	0.000100389	0.011472849
	30-BA	-0.195305	2.548597	1.509524	-0.49697	2.543233	1.498988	0.011923876	0.001685389	0.012042399
	35-UR	-0.145257	2.450323	1.409662	-0.495002	2.542593	1.499517	0.011778277	0.000282611	0.011781667
	40-LL	-0.095363	2.81858	1.639607	-0.496078	2.542314	1.500461	0.011142625	0.000793389	0.011170835
	45-UR	-0.045349	2.388569	1.359507	-0.4949	2.542814	1.500201	0.01108076	0.000384611	0.011087433
	45-CE	-0.045362	2.576667	1.550397	-0.496992	2.543795	1.500417	0.010398107	0.001707389	0.010537353
	average							0.011523947	0.002026972	0.011886668
End Effector Goal Position					-0.495285	2.54874	1.509564			

Figure 4-22 lists the end effector positioning average errors based on the three approaches tested. These are represented as error in the X direction, error in the Y-Z plane, and the full error in X-Y-Z 3D space.

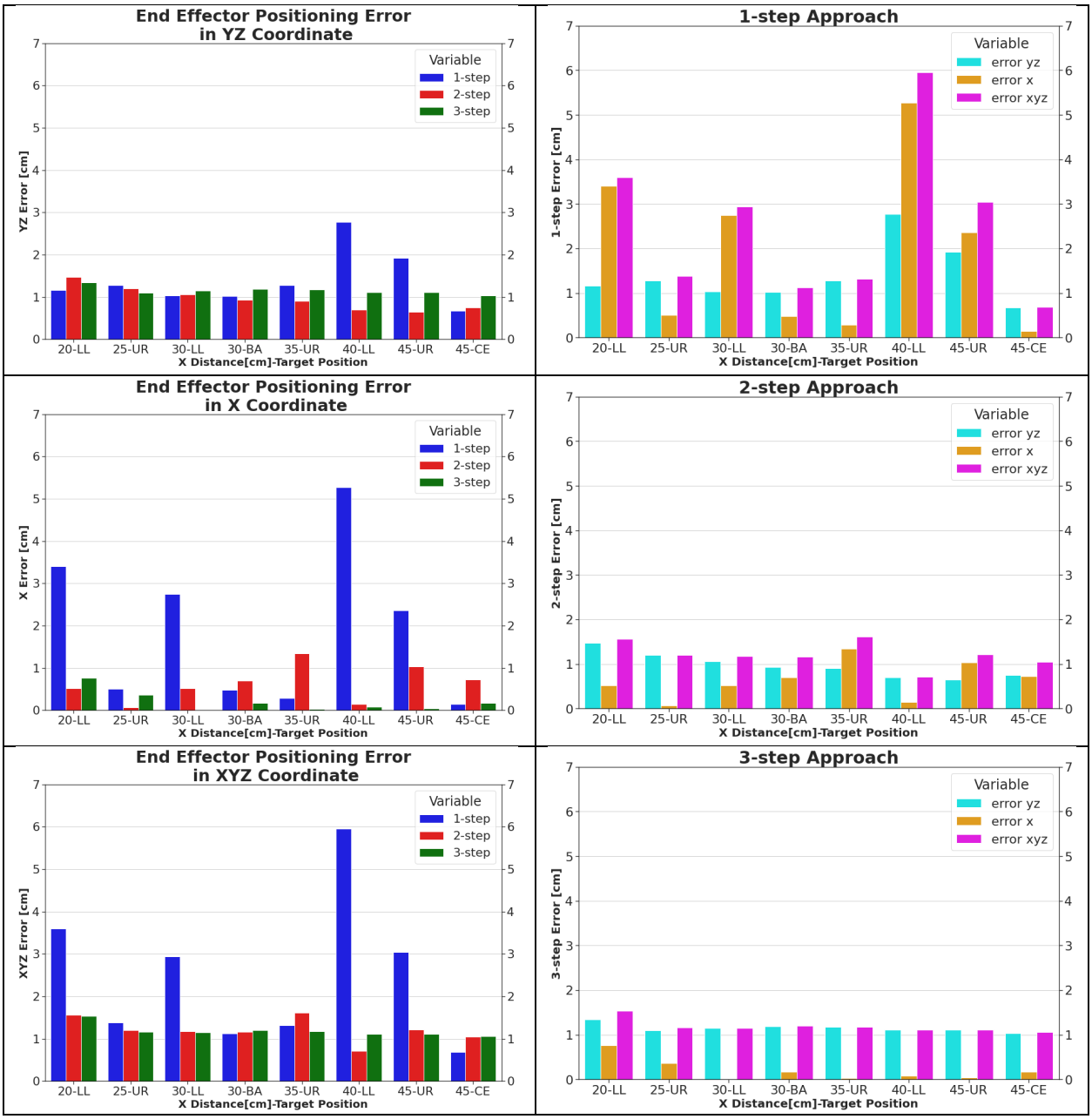


Figure 4-22 End effector positioning error (charts on left by coordinates, charts on right by approach type)

Figure 4-23 depicts the end effector positioning average error based on the three approaches tested, represented as error in the Y-Z plane, error in the X direction, and the full error in X-Y-Z 3D space. It can be seen that the 2-step approach significantly reduces error (both in X and Y-Z) relative to the 1-step approach, and that the 3-step approach further reduces the error in [x].

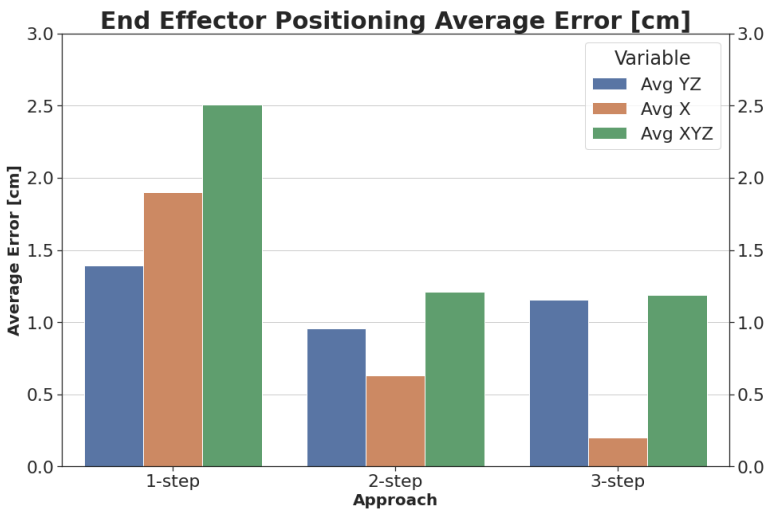


Figure 4-23 End effector average positioning error (cm)

4.4.2. End Effector Time and Space Analysis

Figure 4-24 contains two views of the same graph which depicts the end effector trace in 3D world space. The orange dot shows the initial position of the end effector, and the blue dot shows the ideal final position of the end effector. The three lines show the end effector path for three distinct approaches (1-step, 2-step, 3-step).

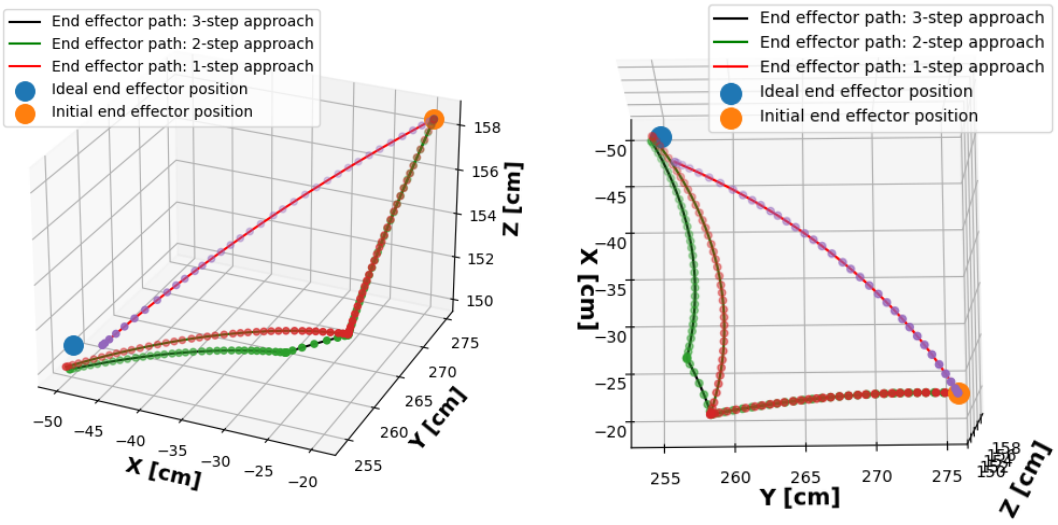


Figure 4-24 Comparing three different approaches in 3D

Figure 4-25 shows the time analysis of the end effector positioning error. The trade-off of the 1-step method rather than the multi-step methods is the settling time of the robot's end effector. It can be seen that the 1-step approach is settled faster than the 2-step, and the 2-step approach faster than the 3-step.

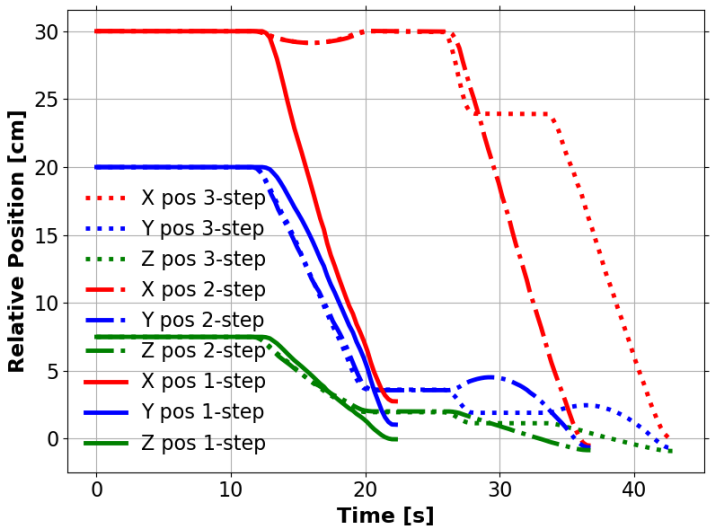


Figure 4-25 Time analysis of the end effector positioning error

The final result is the integration of the task planner, image analyzer, and auto position controller into the simulation. The following depicts an example scenario (illustrated in Figure 4-26) demonstrating this integration:

1) Initialize the robot system.
2) Move the rail to near the 3D printer.
3) Move the end effector to near the 3D printer plate.
4) Perform image analysis and position control to center the object in the camera view.
5) Perform image analysis and position control to further center the object in the camera view and move the end effector in the X direction to just outside the object's gripping target.
6) Execute a predefined "grasping" movement that embraces the object's gripping target in the gripper.

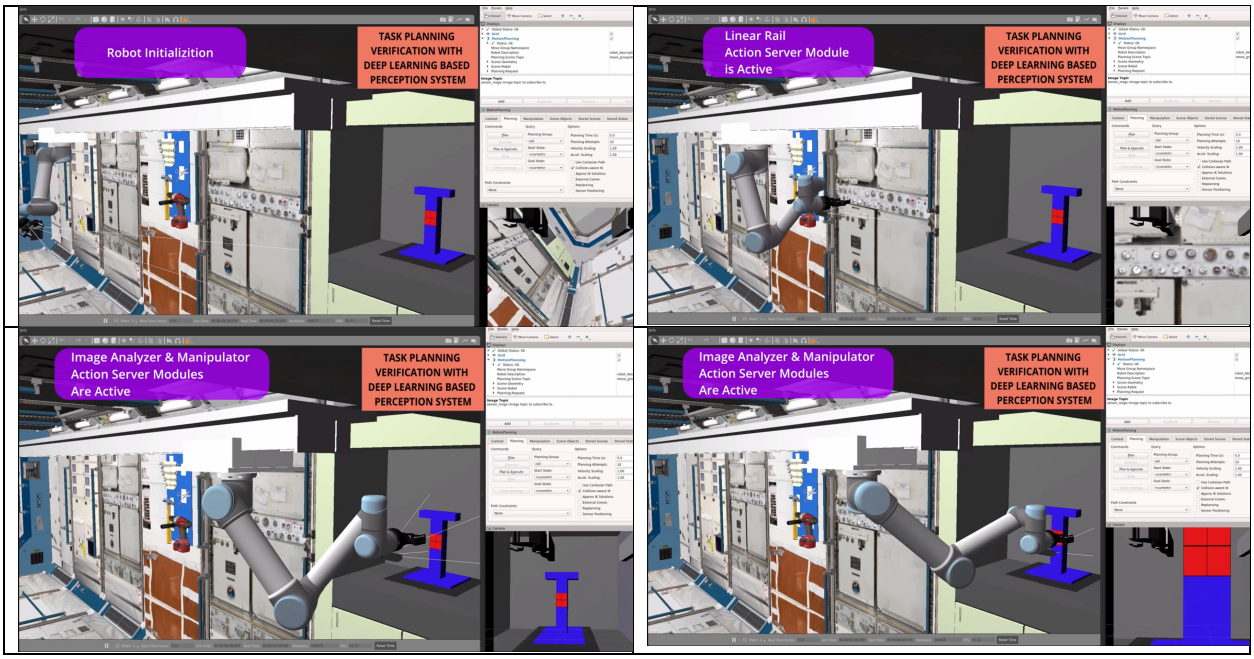


Figure 4-26 Four frames from an integrated simulation example

CHAPTER 5

5. Conclusions and Future Work

CHAPTER 5

5.1. Introduction

The main objective of this dissertation was to design and develop AI-based Robotic task planning and guidance systems for unstructured environments. We have developed two completely independent task planning and guidance systems for two separate applications. The first system is for roadway robotic crack sealing and cleaning. This system uses a vehicle-based simple robotic arm with an eye-to-hand vision configuration. Second, we developed a guidance system for the application of on-board space manufacturing. This system consists of a universal robot arm with 6 degrees of freedom attached to a linear rail providing a seventh degree of freedom. The depth camera is attached to the gripper of the robot (eye-in-hand configuration). Both systems developed share common architectural concepts.

Figure 5-1 shows the sensor-directed task planning architecture developed in this dissertation. In this architecture, the sensor and perception of the robot agent is separated from the world. This separation as shown clears the path for exact sensory information that can be used by the robot or to calibrate the model of the world. The perceptual system can be very sophisticated and incorporate several advanced data sources. The next element of this architecture is the task controller. The task controller is where the agent-based system receives the featured data from the perception system, aligns it with the current plan, and generates an action. The actions, once finally executed, will change the state of the world. The task controller is designed to behave reactively, which means that it helps to accomplish the task and compensates for local inconsistencies. Given the example of a robot gripper grabbing an object, if the relative position of the object and the robot has slightly changed, the task controller takes care of the position change.

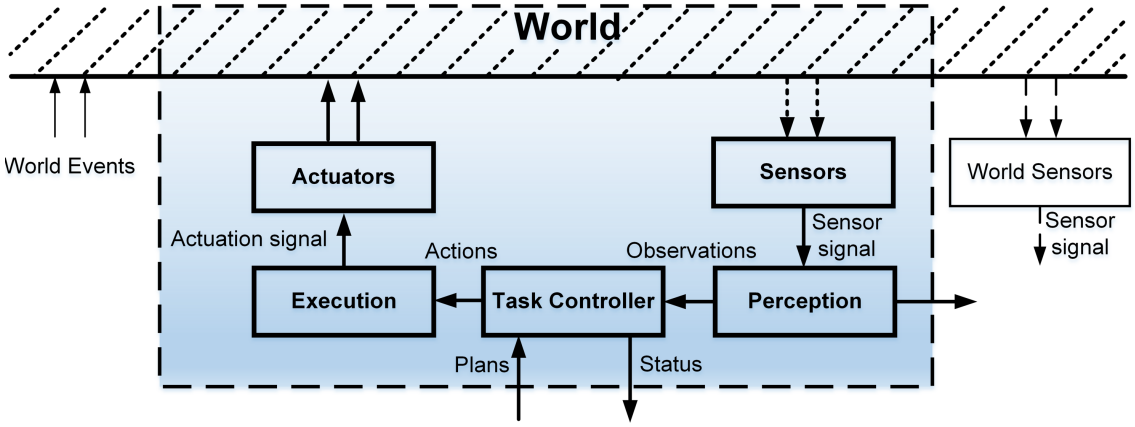


Figure 5-1 Sensor-directed task planning

Once the sensor-directed task planning is developed for every individual robot agent, then a central task planner can control a network of robots that interact with each other as conceptualized in Figure 5-2. Under the policy of the central task planning, each robot agent interacts with another agent and with the world. Each robot also receives its plan from the planner. In this scenario, every robot is helping to build the model of the world that is ultimately used by the planner to accomplish the goals of the operations.

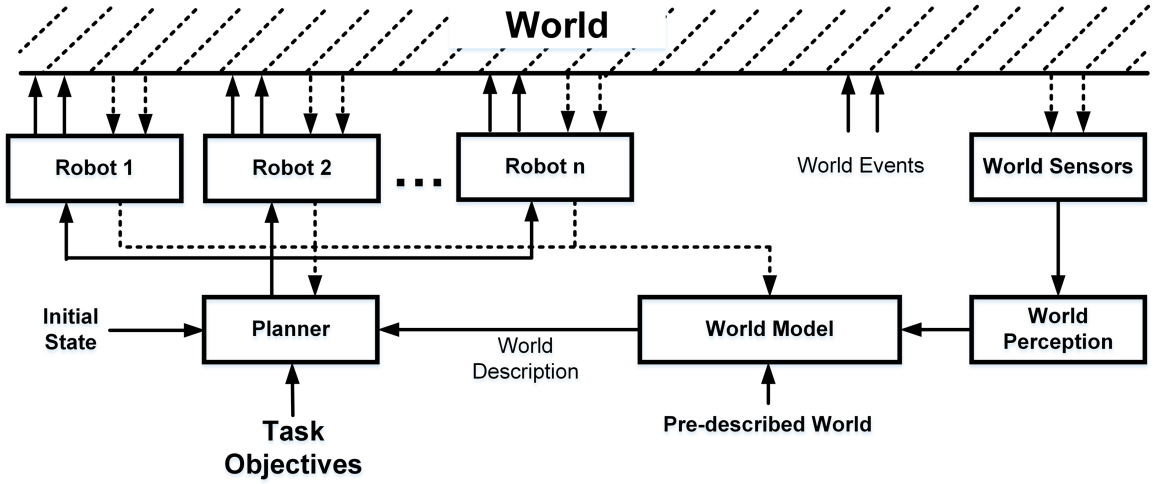


Figure 5-2 The conceptual sensor-directed task planning by separating robot agents from the world

5.2. Robotic Task Planning for Crack Detection and Sealing/Routing

For robotic application in roadway edge crack detection and sealing/routing, this dissertation developed methods to minimize the need of an operator or to make the job of the operator less cumbersome for these unstructured maintenance tasks. On roadways, the operator can still supervise the procedure and override the command control, but the reactive tracking control system is capable of handling and maneuvering the end effector over the pavement crack for sealing or routing and cleaning operations. This real time system is expected to be effective if it is deployed on a maintenance vehicle, relying on the driver to control the vehicle so that it would remain close and parallel to the shoulder where the roadway edge crack exists. Future work in this area can include using a robotic arm with more degrees of freedom and a longer reach to compensate for driver deviations from the roadway edge crack.

The task planning system developed for roadway crack sealing/routing has several advanced capabilities, including crack detection, three-mode task planning, and manipulator control. The system task planning provides operational flexibility by tracking and controlling in automatic, semi-automatic, and manual modes. The system's image processing technology is based on a deep convolutional neural network, one of the most powerful machine vision technologies, allowing the system to identify the crack in the image by analyzing real-time image frames.

The automated tracking technique directs the tool head along the crack's edge. The tracking procedure begins with capturing digital images of the tool head's environment. A straight-line path representing the crack edge is identified by digitally processing the image with an edge detection method. The tracking algorithm calculates the offset distance between the current end-tool/router position and the crack edge after extrapolating the edge path to the tool head.

This offset value is then converted to a target tool head position. A joystick controller provides the system operator with manual lateral positional control in order for tracking to begin and for crack edge inconsistencies to be alleviated in manual or supervisory control operations.

System image processing utilizes the most advanced machine vision filtering technology, a deep convolutional neural network (DCNN). This filter determines the position of the crack inside the image by analyzing real-time frames. Unlike traditional image processing techniques using a preset filtering parameter DCNN, the filtering parameters are optimized using regression with labeled images that we collected from real roadway footage.

The system task planner uses the target position resulting from image processing to control the end-effector's position. The operator can override this position through a joystick input. However, if the system is set to auto-guiding mode – which is the default mode – then there is no need for the operator to intervene. The crack-sensing position feedback is used to control the position of the linear robotic manipulator. For this study, we limited the robotic system to a single linear-prismatic-manipulator; however, any sophisticated robotic arm with multiple degrees of freedom can be deployed in the future. To command the position, we used a simple PID controller rather than a sophisticated control strategy. A concept for future implementation of this technology is depicted in Figure 5-3a. In this Figure, a robotic arm is integrated into a maintenance vehicle equipped with the crack detection system. Figure 5-3b depicts the data flow structure for this system consisting of the control computer, the display and the joystick for accommodating different levels of autonomy from manual control to semi-automatic supervisory control and autonomous operations.

a) b)

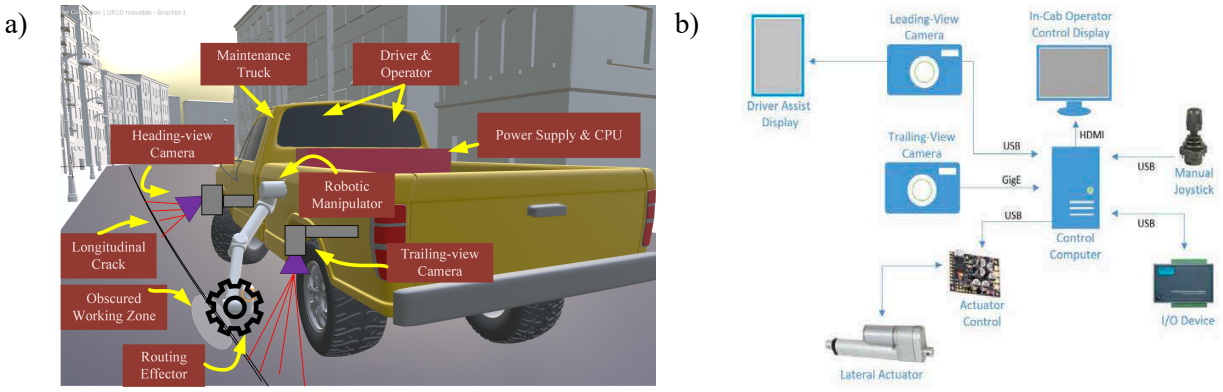


Figure 5-3 Automated longitudinal crack guidance system a) Physical structure b) Data flow structure

5.3. Sensor-guided Robotic Task Planning for Interaction with On-board 3D Printing on a Space Habitat

In this dissertation, we developed an on-board sensor-guided robotic system for fine motion planning in space where the position of objects may be perturbed or new parts are fabricated on-board. This system has a vision-based task planning and perceptual system to direct the manipulator control. The perceptual system includes a visual filtering technique to extract feature points from the object to be handled by the robot. The filter identifies the feature points in the object frame and analyzes them in real time. Image processing filters utilize a DCNN YOLO filter to identify the markers on the object. The output of this filter allows tracking the object.

The system developed also includes a robust simulation environment that can model multi-agent systems. The simulation system allows modeling a metal 3D printer and payload rack, and RGB and RGBD sensors for machine vision. The simulation system also incorporates a physics engine for off-line task planning, verification, and programming of smart habitat robotic tasks. Physics-based simulations can play a crucial role in robotic task verification in mission-critical operations and provide an interface between human and robot teams. The system can serve as a testbed for robotic task planning and execution.

The system developed would benefit from a number of future enhancements. Adding situational awareness (e.g., detection of anomalous sensor data and other unexpected circumstances) and an AI-powered decision-making interface would allow astronauts to efficiently deal with on-board situations that need attention. The task planner and control manager system could be expanded to handle additional intelligent robotic systems and tasks, including boundary enforcement. The image processing system could also be enhanced to accommodate calibration of the CAD models used in the simulation. In addition, ROS2 could be integrated into the system via a ROS bridge, allowing more distributed data service capabilities.

A schematic view of all the components and layers of an ideal task planning and simulation system is depicted in Figure 5-4 An ideal. This system's robotic advance task planner could handle automation of various tasks. The system would process sensor data output and develop and execute a task plan. The plans could also be overridden by the on-board astronauts. The system default, however, would be set to auto mode without any need for the astronaut to intervene, allowing the task to be completed.

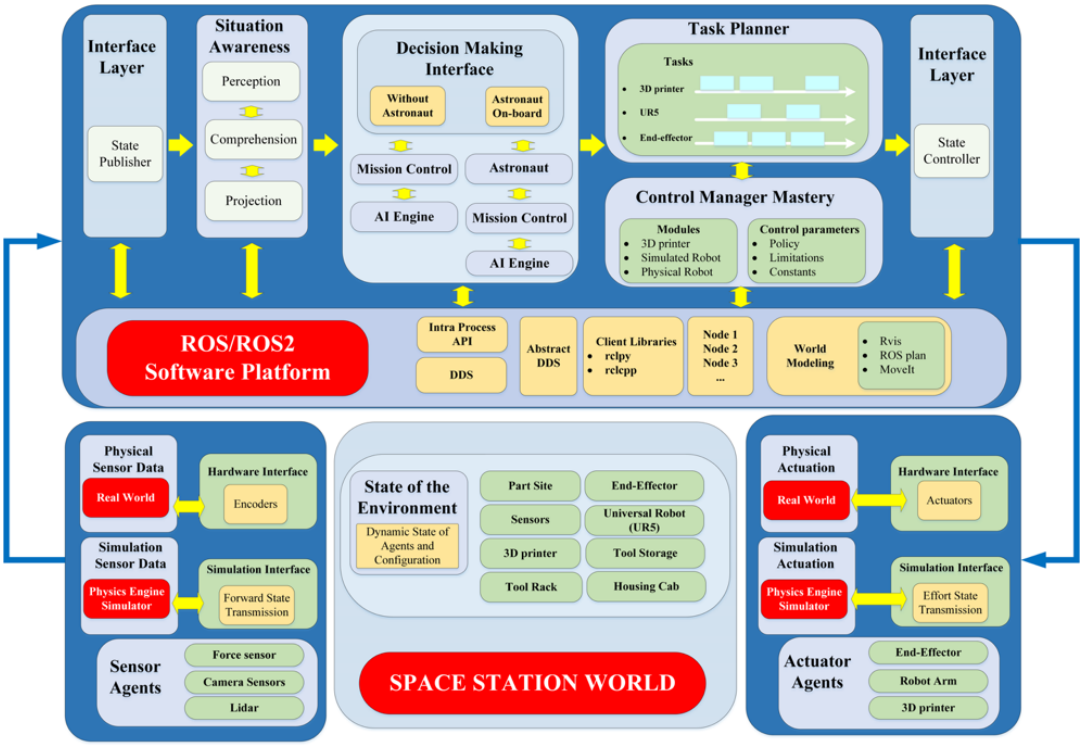


Figure 5-4 An ideal architecture platform for task planning and simulation using ROS and Gazebo

List of References

[1] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning and Acting*. Cambridge University Press, 2016.

[2] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS: a practical introduction to the Robot Operating System*. O'Reilly Media, Inc., 2015.

[3] G. Best, "Planning Algorithms for Multi-Robot Active Perception," PhD Doctorate, The University of Sydney, 2019. [Online]. Available: http://hdl.handle.net/2123/19781

[4] D. Bennett and S. A. Velinsky, "Development of the Sealzall Machine: upgrade to the TTLS (pavement crack sealer).," California. Dept. of Transportation. Division of Research and Innovation, 2009.

[5] M. M. Trivedi and C. Chen, "Sensor-Driven Intelligent Robotics**This research was supported in part by the DOE's University Program in Robotics for Advanced Reactors (Universities of Florida, Michigan, Tennessee, and Texas and the Oak Ridge National Laboratory) under grant DOE-DE-FG02-86NE37968.," in *Advances in Computers*, vol. 32, M. C. Yovits, Ed. Elsevier, 1991, pp. 105–148. doi: 10.1016/S0065-2458(08)60246-6.

[6] R. D. Hale, M. Rokonuzzaman, and R. G. Gosine, "Control of mobile robots in unstructured environments using discrete event modeling," in *Mobile Robots XIV*, Nov. 1999, vol. 3838, pp. 275–282. doi: 10.1117/12.369263.

[7] B. Brunner, K. Arbter, and G. Hirzinger, "Task-directed programming of sensor-based robots," in *Intelligent Robots and Systems*, 1995, pp. 387–400.

[8] R. D. Hale, M. Rokonuzzaman, and R. G. Gosine, "Control of mobile robots in unstructured environments using discrete event modeling," in *Mobile Robots XIV*, Nov. 1999, vol. 3838, pp. 275–282. doi: 10.1117/12.369263.

[9] G. Wittenberg, "Developments in offline programming: an overview," *Ind. Robot Int. J.*, 1995.

[10] F. Leali, M. Pellicciari, F. Pini, G. Berselli, and A. Vergnano, "An offline programming method for the robotic deburring of aerospace components," in *International Workshop on Robotics in Smart Manufacturing*, 2013, pp. 1–13.

[11] Z. Pan, J. Polden, N. Larkin, S. van Duin, and J. Norrish, "Automated offline programming for robotic welding system with high degree of freedoms," in *Advances in computer, communication, control and automation*, Springer, 2011, pp. 685–692.

[12] J. Polden, Z. Pan, N. Larkin, S. Van Duin, and J. Norrish, "Offline programming for a complex welding system using DELMIA automation," in *Robotic Welding, Intelligence and Automation*, Springer, 2011, pp. 341–349.

[13] M. H. Luque and A. W. Stokes, "Fishing Behaviour of Alaska Brown Bear," *Bears Their Biol. Manag.*, vol. 3, p. 71, 1976, doi: 10.2307/3872756.

[14] "(31) Pinterest," *Pinterest*. https://www.pinterest.com/pin/328410997802615113/ (accessed May 13, 2021).

[15] A. U. Frank, S. Bittner, and M. Raubal, "Spatial and cognitive simulation with multi-agent systems," in *International Conference on Spatial Information Theory*, 2001, pp. 124–139.

[16] "Definition of MAINTENANCE." https://www.merriam-webster.com/dictionary/maintenance (accessed Aug. 11, 2021).

[17] *Read "The Competitive Edge: Research Priorities for U.S. Manufacturing" at NAP.edu*. doi: 10.17226/1618.

[18] L. E. Parker and J. V. Draper, "Robotics applications in maintenance and repair," *Handb. Ind. Robot.*, vol. 2, pp. 1023–1036, 1998.

[19] R. J. Firby, "Task directed sensing," in *Sensor Fusion II: Human and Machine Strategies*, 1990, vol. 1198, pp. 480–489.

[20] D. Marr, “Vision: A computational investigation into the human representation and processing of visual information,” 1982.
[21] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*. Elsevier, 2004.
[22] R. Brooks, “A robust layered control system for a mobile robot,” *IEEE J. Robot. Autom.*, vol. 2, no. 1, pp. 14–23, Mar. 1986, doi: 10.1109/JRA.1986.1087032.
[23] P. E. Agre and D. Chapman, “Pengi: An Implementation of a Theory of Activity.,” in *AAAI*, 1987, vol. 87, no. 4, pp. 286–272.
[24] R. Knight, F. Fisher, T. Estlin, B. Engelhardt, and S. Chien, “Balancing deliberation and reaction, planning and execution for space robotic applications,” in *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*, 2001, vol. 4, pp. 2131–2139.
[25] J. T. Doerflinger, T. Martiny-Huenger, and P. M. Gollwitzer, “Planning to deliberate thoroughly: if-then planned deliberation increases the adjustment of decisions to newly available information,” *J. Exp. Soc. Psychol.*, vol. 69, pp. 1–12, 2017.
[26] A. J. Briggs and B. R. Donald, “Automatic sensor configuration for task-directed planning,” in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, 1994, pp. 1345–1350.
[27] M. Quigley *et al.*, “ROS: an open-source Robot Operating System,” in *ICRA workshop on open source software*, 2009, vol. 3, no. 3.2, p. 5.
[28] D. Bennett and S. A. Velinsky, “Deployment Support and Caltrans’ Implementation of the Sealzall Machine,” 2014. [Online]. Available: http://ahmct.ucdavis.edu/pdf/UCD-ARR-13-06-30-03.pdf
[29] W. Rawat and Z. Wang, “Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review,” *Neural Comput.*, vol. 29, no. 9, pp. 2352–2449, Jun. 2017, doi: 10.1162/neco_a_00990.
[30] K. Gopalakrishnan, “Deep learning in data-driven pavement image analysis and automated distress detection: A review,” *Data*, vol. 3, no. 3, p. 28, 2018.
[31] N. Chernov, *Circular and linear regression: Fitting circles and lines by least squares*. CRC Press, 2010.
[32] R. M. Hirsch and E. J. Gilroy, “METHODS OF FITTING A STRAIGHT LINE TO DATA: EXAMPLES IN WATER RESOURCES 1,” *JAWRA J. Am. Water Resour. Assoc.*, vol. 20, no. 5, pp. 705–711, 1984.
[33] *tensorflow/models*. tensorflow, 2021. Accessed: Jun. 01, 2021. [Online]. Available: https://github.com/tensorflow/models/blob/0c9253b4a0b34935cf78bd13e6520bbeee2f5f92/research/object_detection/g3doc/tf1_detection_zoo.md
[34] A. Owens and O. De Weck, “Systems Analysis of In-Space Manufacturing Applications for the International Space Station and the Evolvable Mars Campaign,” in *AIAA SPACE 2016*, 2016, p. 5394.
[35] W. Cirillo, G. Aaseng, K. Goodliff, C. Stromgren, and A. Maxwell, “Supportability for beyond low earth orbit missions,” in *AIAA SPACE 2011 Conference & Exposition*, 2011, p. 7231.
[36] K. Hurlbert, B. Bagdigian, C. Carroll, A. Jeevarajan, M. Kliss, and B. Singh, “Human Health, Life Support and Habitation Systems,” *Natl. Aeronaut. Space Adm. NASA Wash. DC U. S. Am.*, 2012.
[37] J. Agte, N. Borer, and O. de Weck, “Design of long-endurance systems with inherent robustness to partial failures during operations,” 2012.
[38] H. W. Jones, E. W. Hodgson, and M. H. Kliss, “Life support for deep space and Mars,” 2014.
[39] B. Piascik, J. Vickers, D. Lowry, S. Scotti, J. Stewart, and A. Calomino, “Materials, structures, mechanical systems, and manufacturing roadmap,” *NASA TA*, pp. 12–2, 2012.
[40] W. R. Longhurst *et al.*, “Development of friction stir welding technologies for in-space manufacturing,” *Int. J. Adv. Manuf. Technol.*, vol. 90, no. 1–4, pp. 81–91, 2017.
[41] J. Gregg, “The Industrialization of Space,” in *The Cosmos Economy*, Springer, 2021, pp. 133–134.

[42] "Zero-G Space-Based Factories May be The Future of Our World," *Futurism*. https://futurism.com/neoscope/zero-g-space-based-factories-may-be-the-future-of-our-world (accessed Aug. 21, 2021).
[43] D. E. Bernard *et al.*, "Remote agent experiment ds1 technology validation report," *Ames Res. Cent. JPL*, pp. 8–9, 2000.
[44] L. P. Patterson, "On-orbit maintenance operations strategy for the international space station—concept and implementation," in *AIP Conference Proceedings*, 2001, vol. 552, no. 1, pp. 139–146.
[45] N. Sreekanth, A. Dinesan, A. R. Nair, G. Udupa, and V. Tirumaladass, "Design of robotic manipulator for space applications," *Mater. Today Proc.*, vol. 46, pp. 4962–4970, 2021.
[46] M. Hiltz, C. Rice, K. Boyle, and R. Allison, "Canadarm: 20 years of mission success through adaptation," 2001.
[47] R. E. Arvidson *et al.*, "Opportunity Mars Rover mission: Overview and selected results from Purgatory ripple to traverses to Endeavour crater," *J. Geophys. Res. Planets*, vol. 116, no. E7, 2011.
[48] S. Aswath *et al.*, "Design and Development of an Intelligent Rover for Mars Exploration," 2016.
[49] "Meet the Astrobees! These Tiny, Cube-Shaped Robots Have Arrived in Space | Space." https://www.space.com/astrobee-robots-on-space-station.html (accessed May 06, 2021).
[50] T. Reichhardt, "Robot Helpers Are Coming to the Space Station," *Air & Space Magazine*. https://www.airspacemag.com/space/space-station-robots-180967704/ (accessed May 06, 2021).
[51] M. Johnson, "Space Station Robotic Arms Have a Long Reach," *NASA*, Mar. 18, 2019. http://www.nasa.gov/mission_pages/station/research/news/b4h-3rd/hh-robotic-arms-reach (accessed May 06, 2021).
[52] "Dextre (Special Purpose Dexterous Manipulator)," *Supercluster*. https://www.supercluster.com/astronauts/dextre-(special-purpose-dexterous-manipulator) (accessed May 06, 2021).
[53] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, 2004, vol. 3, pp. 2149–2154.
[54] W. Qian *et al.*, "Manipulation task simulation using ROS and Gazebo," in *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*, 2014, pp. 2594–2598.
[55] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS: a practical introduction to the Robot Operating System*. O'Reilly Media, Inc., 2015.
[56] "MoveIt Motion Planning Framework." https://moveit.ros.org/ (accessed Oct. 20, 2021).
[57] "Home," *OpenCV*. https://opencv.org/ (accessed Oct. 20, 2021).
[58] A. D. Wilson, "Using a depth camera as a touch sensor," in *ACM international conference on interactive tabletops and surfaces*, 2010, pp. 69–72.
[59] "OpenCV: Deep Neural Network module." https://docs.opencv.org/4.2.0/d6/d0f/group__dnn.html#ga29f34df9376379a603acd8df581ac8d7 (accessed Jan. 09, 2022).
[60] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *ArXiv Prepr. ArXiv180402767*, 2018.
[61] D. J. Bora, "A novel approach for color image edge detection using multidirectional Sobel filter on HSV color space," *Int J Comput Sci Eng*, vol. 5, no. 2, pp. 154–159, 2017.
[62] F. Tao, M. Zhang, and A. Y. C. Nee, *Digital twin driven smart manufacturing*. Academic Press, 2019.

Appendix A: Machine Vision GUI

In the crack cleaning/sealing machine vision application, the operator is able to modify the image processing parameters in real time and observe their effect. Figure A-1 shows an application screenshot annotated with the descriptions of each of these parameters.

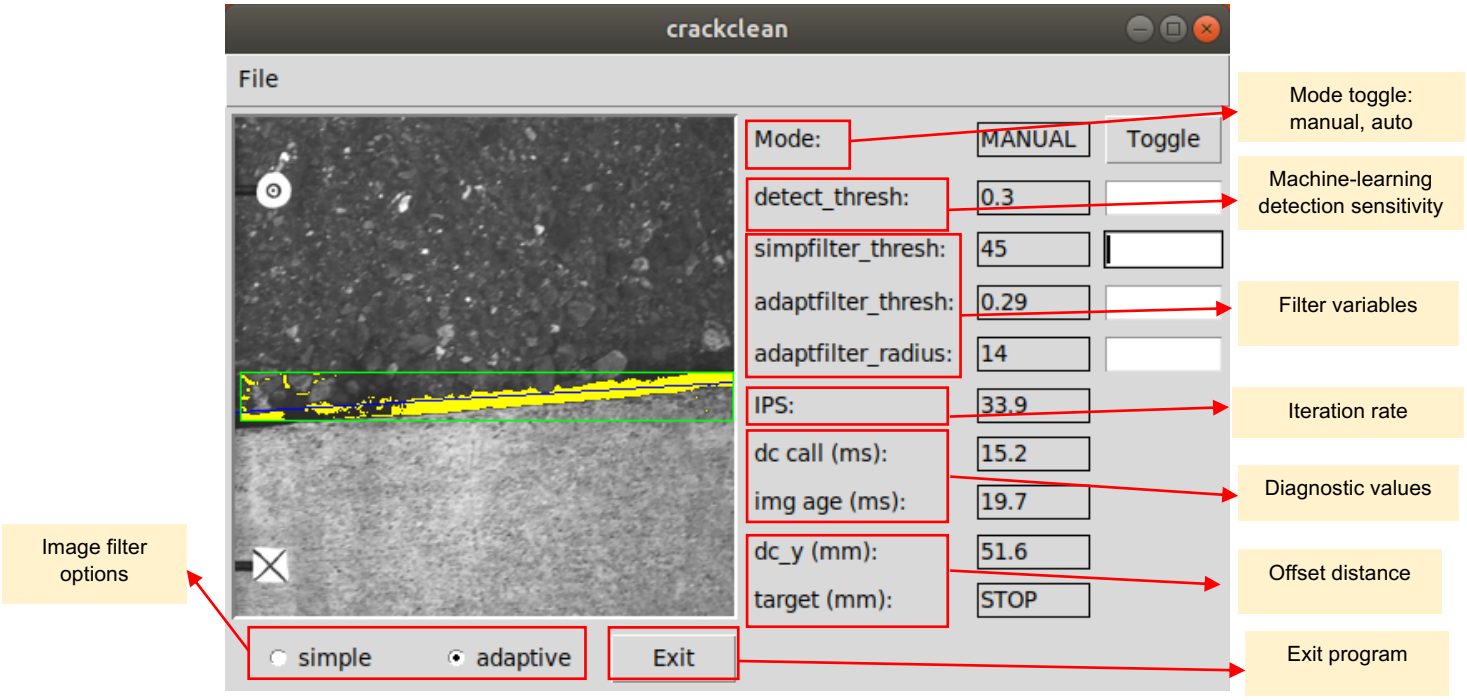


Figure A-1 Machine vision application for the operator

Appendix B: Crack Clean Testbed Specifications

Table B-1 lists specifications of the major components of the crack clean testbed described in Section 3.5.

Table B-1 Crack clean testbed specifications

Component	Brand	Description
Laptop	Dell Precision, M 4700	CPU Core i7
Camera	FLIR Blackfly	CCD, 52 fps, 4k
Tensor Processing Unit	Progressive Automation	USB port, Single Axis
Linear Actuator	Megatron	2 inch/sec, 12” stroke

Appendix C: Image Processing HSV Filter

Early in the research, a proof-of-concept object location algorithm (Figure C-1) was developed that worked by performing hue, saturation, and value (HSV) range filtering [61]. Image pixels within a specified HSV range were processed in order to estimate the position of an object target marker.

```
1.  void ImageAnalyzer::hsv_filter(Mat& img) {
2.      float b_thresh = 170.0;
3.      float b_window = 1;
4.      int low_H = (int)((b_thresh-b_window) * (255.0/360.0));
5.      int high_H = (int)((b_thresh+b_window) * (255.0/360.0));
6.      int low_S = (int)(60.0 * (255.0/100.0));
7.      int high_S = (int)(100.0 * (255.0/100.0));
8.      int low_V = (int)(60.0 * (255.0/100.0));
9.      int high_V = (int)(100.0 * (255.0/100.0));
10.     std::cout << "lo/hi H: " << low_H << "," << high_H << std::endl;
11.     Mat img_HSV, img_filter;
12.     // Convert from BGR to HSV colorspace
13.     cvtColor(img, img_HSV, COLOR_BGR2HSV);
14.     // Detect the object based on HSV Range Values
15.     inRange(img_HSV, Scalar(low_H, low_S, low_V),
16.         Scalar(high_H, high_S, high_V), img_filter);
17.     int hot_cnt = 0;
18.     int min_c = INT_MAX;
19.     int max_c = INT_MIN;
20.     int min_r = INT_MAX;
21.     int max_r = INT_MIN;
22.     for (int r=0; r<img_filter.rows; ++r) {
23.         for (int c=0; c<img_filter.cols; ++c) {
24.             int val = (int)(img_filter.at<uchar>(r,c));
25.             if (val > 0) {
26.                 hot_cnt++;
27.                 if (c < min_c)
28.                     min_c = c;
29.                 else if (c > max_c)
30.                     max_c = c;
31.                 if (r < min_r)
32.                     min_r = r;
33.                 else if (r > max_r)
34.                     max_r = r;
35.                 // if first is less than something assign it
36.                 // if last is more then replace it
37.             }
38.             //std::cout << val << ",";
39.         }
40.         //std::cout << std::endl;
41.     }
42.     if (hot_cnt > 0) {
43.         std::cout << min_c << std::endl;
44.         std::cout << max_c << std::endl;
45.         std::cout << min_r << std::endl;
46.         std::cout << max_r << std::endl;
47.     }
48.     // display the hsv image
49.     imshow("hello2", img_HSV);
50.     // await user input
51.     waitKey(0);
52.     // display the filter
53.     imshow("filter", img_filter);
54.     // await user input
55.     waitKey(0);
56.     // save a copy
57.     imwrite("outfile.png", img_filter);
58.     return;
59. }
```

Figure C-1 Image analyzer HSV filter

Appendix D: Digital Twin Concepts

Figure D-1 and Figure D-2 illustrate the concept of a digital twin (DT) in the context of a smart station. DT is a digital representation that communicates with the physical object during its lifecycle and provides intelligence for measurement, optimization, and prediction [62]. This application provides communication between the ROS plan, ROS, and Gazebo.

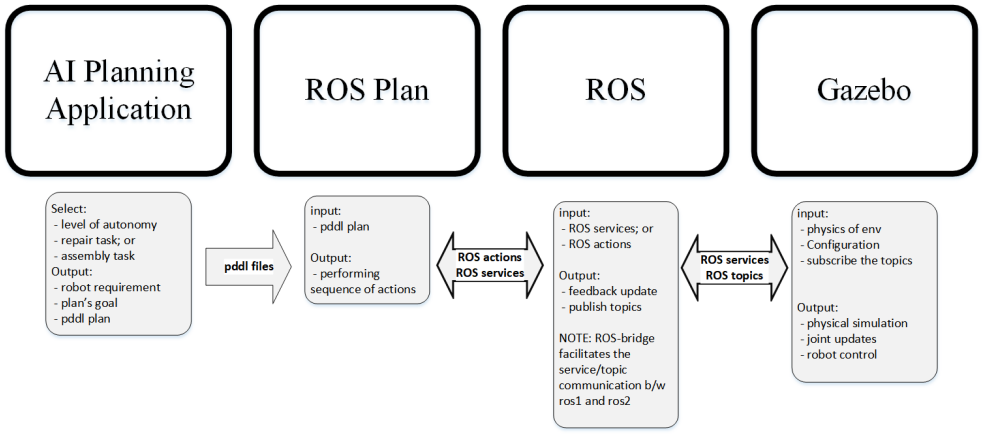


Figure D-1 Block diagram of the AI planning for the digital twin

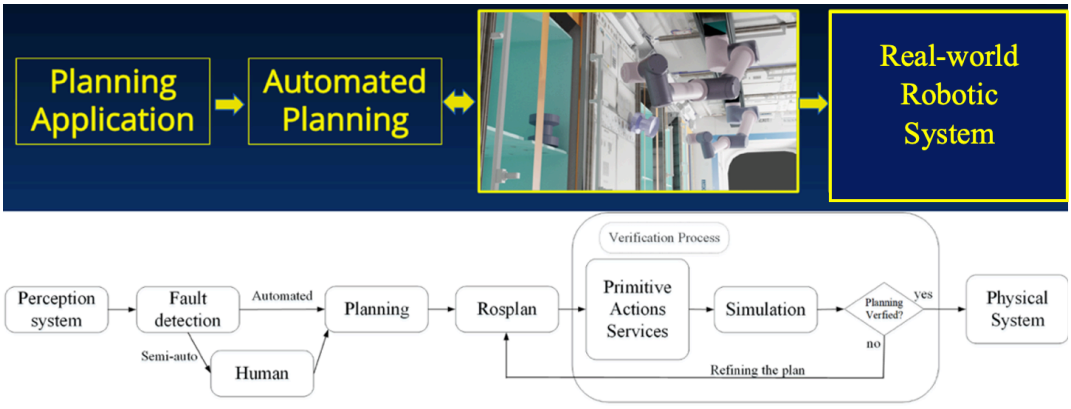


Figure D-2 The digital twin idea for the smart station

Appendix E: Example AI Planning Application

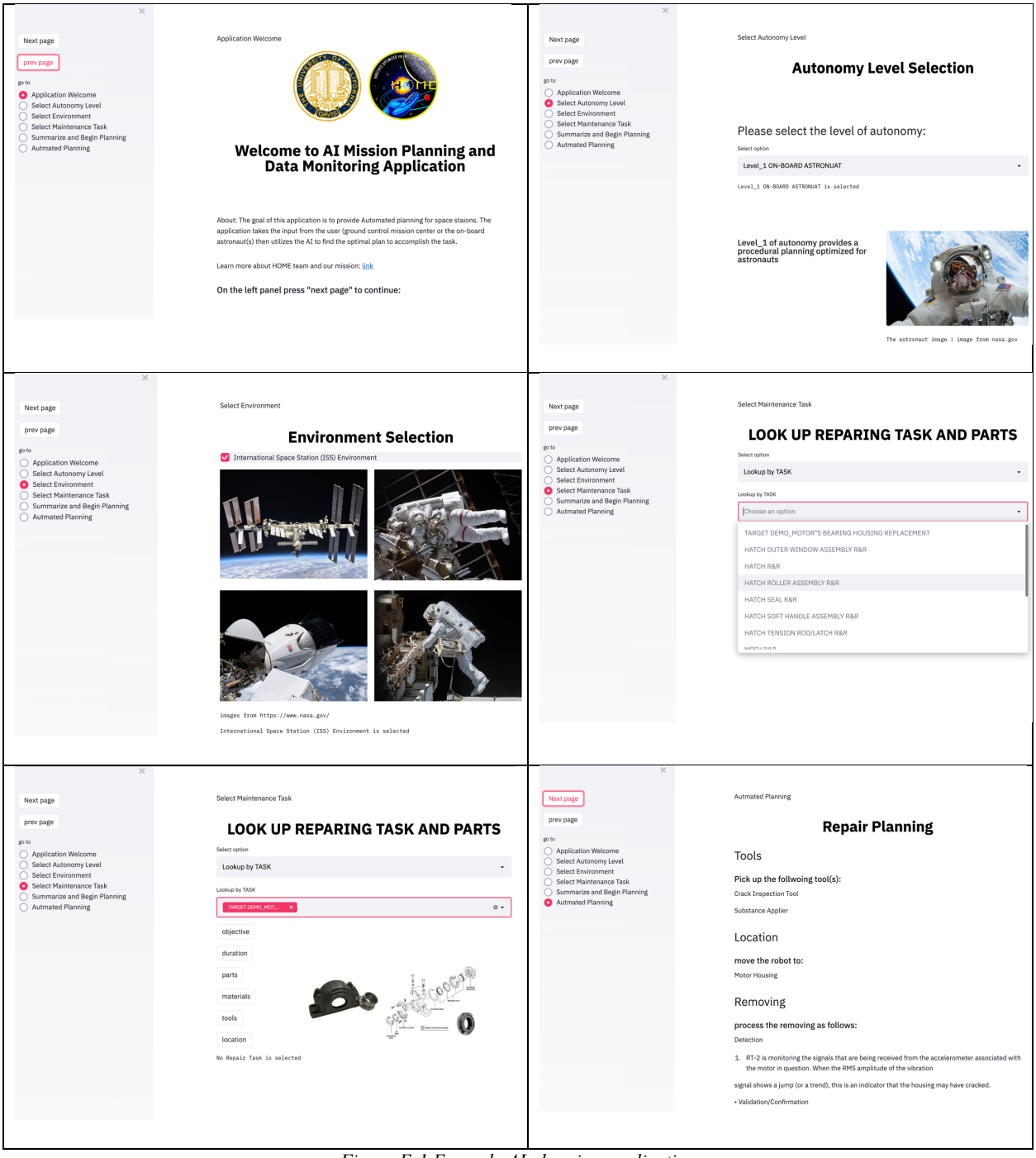


Figure E-1 Example AI planning application

Appendix F: Relative *X* vs. Area and Data Samples for Model Fitting

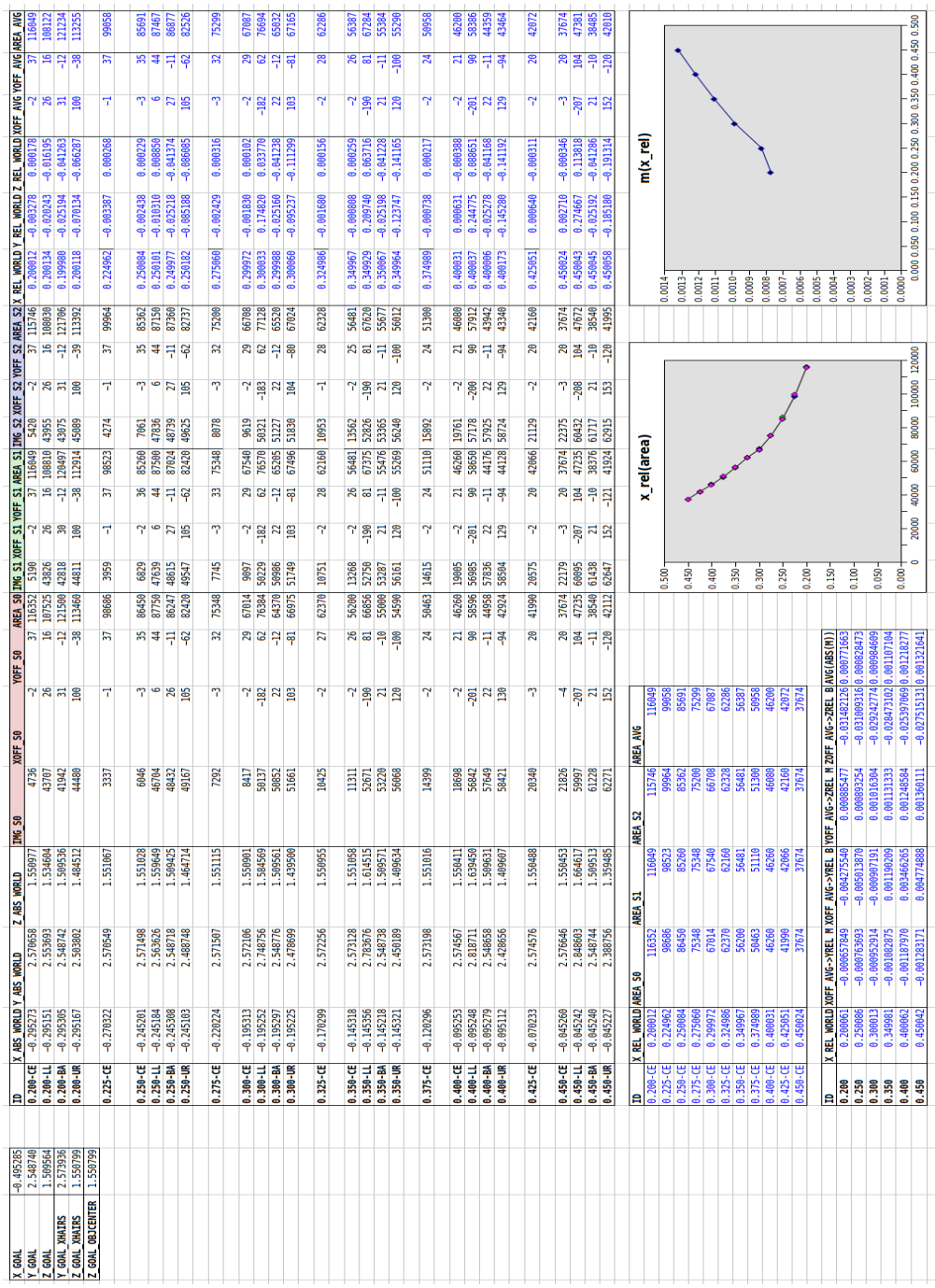


Figure F-1 Relative X vs. area and data samples for model fitting

Appendix G: Model Fitting for Robot Interpolation and Approaching

Figure G-1 details the sampled data used to estimate the relationships between object target offset values in image space and those in world space. Figure G-2 illustrates the linear fit chosen to estimate the coordinate system scaling factor based on the camera's relative x position in world space.

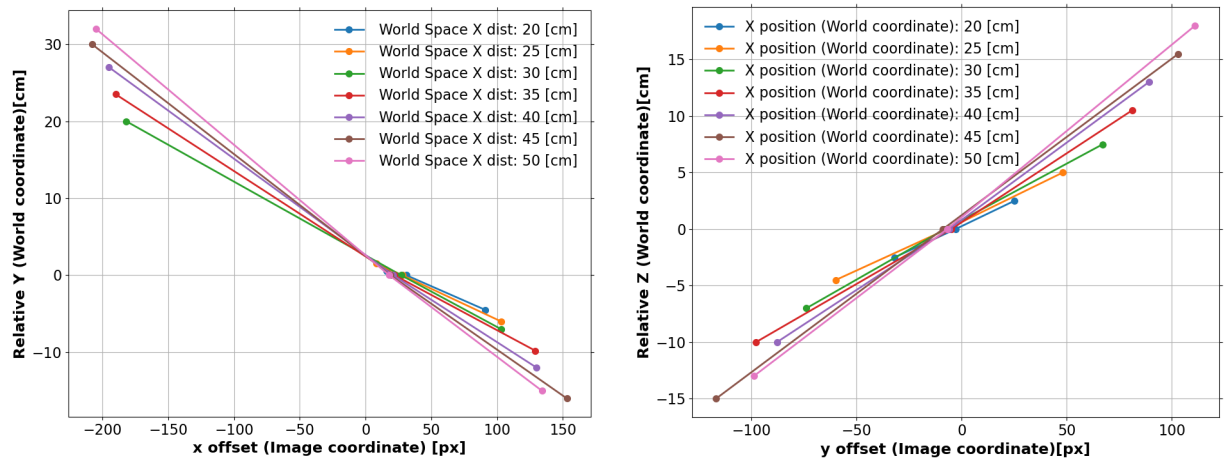


Figure G-1 Offset data sampled from simulation

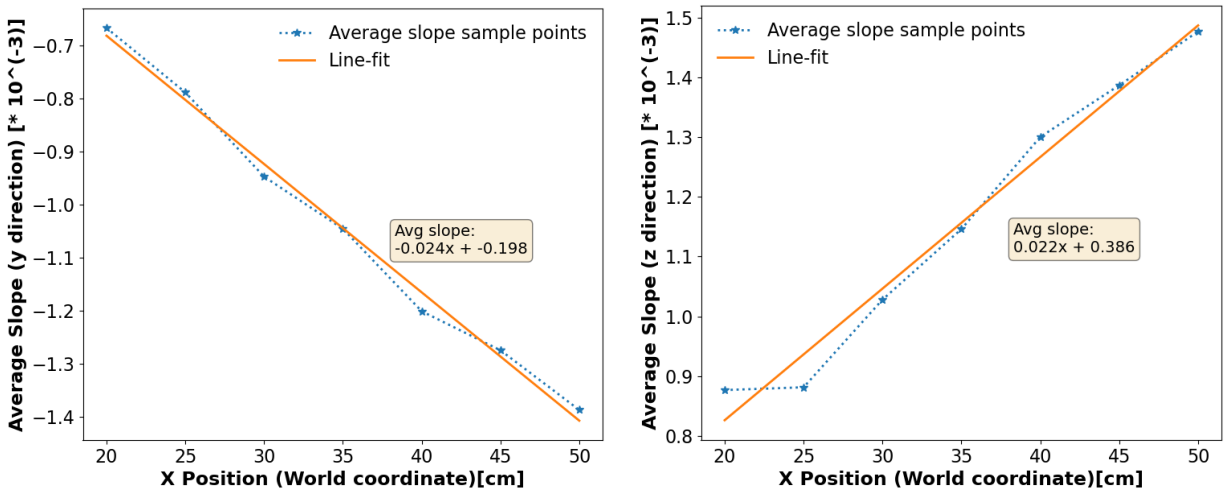


Figure G-2 Curve fitting for scaling factor

Appendix H: Model Setting and Training for YOLO

```
CUDA-version: 11010 (11020), cuDNN: 7.6.5, CUDNN_HALF=1, GPU count: 1
 CUDNN_HALF=1
 OpenCV version: 3.2.0
 0 : compute_capability = 600, cudnn_half = 0, GPU: Tesla P100-PCIE-16GB
net.optimized_memory = 0
mini_batch = 1, batch = 16, time_steps = 1, train = 0
   layer   filters  size/strd(dil)      input                output
   0 Create CUDA-stream - 0
 Create cudnn-handle 0
conv     32       3 x 3/ 1    416 x 416 x   3 ->  416 x 416 x  32 0.299 BF
   1 conv     64       3 x 3/ 2    416 x 416 x  32 ->  208 x 208 x  64 1.595 BF
   2 conv     32       1 x 1/ 1    208 x 208 x  64 ->  208 x 208 x  32 0.177 BF
   3 conv     64       3 x 3/ 1    208 x 208 x  32 ->  208 x 208 x  64 1.595 BF
   4 Shortcut Layer: 1,  wt = 0, wn = 0, outputs: 208 x 208 x  64 0.003 BF
   5 conv    128       3 x 3/ 2    208 x 208 x  64 ->  104 x 104 x 128 1.595 BF
   6 conv     64       1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x  64 0.177 BF
   7 conv    128       3 x 3/ 1    104 x 104 x  64 ->  104 x 104 x 128 1.595 BF
   8 Shortcut Layer: 5,  wt = 0, wn = 0, outputs: 104 x 104 x 128 0.001 BF
   9 conv     64       1 x 1/ 1    104 x 104 x 128 ->  104 x 104 x  64 0.177 BF
  10 conv    128       3 x 3/ 1    104 x 104 x  64 ->  104 x 104 x 128 1.595 BF
  11 Shortcut Layer: 8,  wt = 0, wn = 0, outputs: 104 x 104 x 128 0.001 BF
  12 conv    256       3 x 3/ 2    104 x 104 x 128 ->   52 x  52 x 256 1.595 BF
  13 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  14 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  15 Shortcut Layer: 12,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  16 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  17 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  18 Shortcut Layer: 15,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  19 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  20 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  21 Shortcut Layer: 18,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  22 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  23 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  24 Shortcut Layer: 21,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  25 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  26 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  27 Shortcut Layer: 24,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  28 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  29 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  30 Shortcut Layer: 27,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  31 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  32 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  33 Shortcut Layer: 30,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  34 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
  35 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
  36 Shortcut Layer: 33,  wt = 0, wn = 0, outputs:  52 x  52 x 256 0.001 BF
  37 conv    512       3 x 3/ 2     52 x  52 x 256 ->   26 x  26 x 512 1.595 BF
  38 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  39 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  40 Shortcut Layer: 37,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  41 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  42 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  43 Shortcut Layer: 40,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  44 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  45 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  46 Shortcut Layer: 43,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  47 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  48 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  49 Shortcut Layer: 46,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  50 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  51 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  52 Shortcut Layer: 49,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  53 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  54 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  55 Shortcut Layer: 52,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  56 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  57 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  58 Shortcut Layer: 55,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  59 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  60 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  61 Shortcut Layer: 58,  wt = 0, wn = 0, outputs:  26 x  26 x 512 0.000 BF
  62 conv   1024       3 x 3/ 2     26 x  26 x 512 ->   13 x  13 x1024 1.595 BF
```

```
  63 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  64 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  65 Shortcut Layer: 62,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
  66 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  67 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  68 Shortcut Layer: 65,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
  69 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  70 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  71 Shortcut Layer: 68,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
  72 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  73 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  74 Shortcut Layer: 71,  wt = 0, wn = 0, outputs:  13 x  13 x1024 0.000 BF
  75 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  76 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  77 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  78 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  79 conv    512       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x 512 0.177 BF
  80 conv   1024       3 x 3/ 1     13 x  13 x 512 ->   13 x  13 x1024 1.595 BF
  81 conv     27       1 x 1/ 1     13 x  13 x1024 ->   13 x  13 x  27 0.009 BF
  82 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.00
  83 route  79                                          ->   13 x  13 x 512
  84 conv    256       1 x 1/ 1     13 x  13 x 512 ->   13 x  13 x 256 0.044 BF
  85 upsample                 2x    13 x  13 x 256 ->   26 x  26 x 256
  86 route  85 61                           ->   26 x  26 x 768
  87 conv    256       1 x 1/ 1     26 x  26 x 768 ->   26 x  26 x 256 0.266 BF
  88 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  89 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  90 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  91 conv    256       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x 256 0.177 BF
  92 conv    512       3 x 3/ 1     26 x  26 x 256 ->   26 x  26 x 512 1.595 BF
  93 conv     27       1 x 1/ 1     26 x  26 x 512 ->   26 x  26 x  27 0.019 BF
  94 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.00
  95 route  91                                          ->   26 x  26 x 256
  96 conv    128       1 x 1/ 1     26 x  26 x 256 ->   26 x  26 x 128 0.044 BF
  97 upsample                 2x    26 x  26 x 128 ->   52 x  52 x 128
  98 route  97 36                           ->   52 x  52 x 384
  99 conv    128       1 x 1/ 1     52 x  52 x 384 ->   52 x  52 x 128 0.266 BF
 100 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 101 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
 102 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 103 conv    128       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x 128 0.177 BF
 104 conv    256       3 x 3/ 1     52 x  52 x 128 ->   52 x  52 x 256 1.595 BF
 105 conv     27       1 x 1/ 1     52 x  52 x 256 ->   52 x  52 x  27 0.037 BF
 106 yolo
[yolo] params: iou loss: mse (2), iou_norm: 0.75, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.00
Total BFLOPS 65.326
avg_outputs = 517320
 Allocate additional workspace_size = 52.43 MB
Loading weights from /mydrive/yolov3/training/yolov3-custom_final.weights...
 seen 64, trained: 512 K-images (8 Kilo-batches_64)
Done! Loaded 107 layers from weights-file

 calculation mAP (mean average precision)...
 Detection layer: 82 - type = 28
 Detection layer: 94 - type = 28
 Detection layer: 106 - type = 28
8Can't open label file. (This can be normal only if you use MSCOCO): data/obj/img_no_target_66.txt
Can't open label file. (This can be normal only if you use MSCOCO): data/obj/img_no_target_8.txt
24Can't open label file. (This can be normal only if you use MSCOCO): data/obj/img_no_target_134.txt
28Can't open label file. (This can be normal only if you use MSCOCO): data/obj/img_no_target_26.txt
128
 detections_count = 443, unique_truth_count = 393
class_id = 0, name = obj, ap = 96.17%          (TP = 75, FP = 1)
class_id = 1, name = targ, ap = 99.23%         (TP = 67, FP = 2)
class_id = 2, name = lgrip, ap = 96.33%        (TP = 120, FP = 3)
class_id = 3, name = rgrip, ap = 98.31%        (TP = 118, FP = 3)

 for conf_thresh = 0.25, precision = 0.98, recall = 0.97, F1-score = 0.97
 for conf_thresh = 0.25, TP = 380, FP = 9, FN = 13, average IoU = 88.73 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.975084, or 97.51 %
Total Detection Time: 3 Seconds
```

Figure H-1 Model setting and training for YOLO