Robotic Feedback Loops

Implementing two-way communication in architecturally focused robotic pick and place operations

A 120 point thesis submitted in partial fulfilment of the requirements for the degree of Master of Architecture (Professional)

Victoria University of Wellington, School of Architecture

Harrison Le Fevre

2020

Notes

Unless otherwise stated all images are the authors own work. QR codes have been used throughout this thesis to provide links to additional video based content. These QR codes can be scanned with a QR code reader application. Accessing and watching these videos is essential in understanding the research.

The mission of the Advanced Manufacturing and Prototyping for Design research lab is to investigate and define innovative techniques and methods of construction applicable to the building sector through the use of contemporary tools of design, fabrication, and manufacturing.

Website: ampd.center

Instagram: @ampdlabnz

Abstract

The use of robots in the fabrication of complex architectural structures is increasing in popularity. However, architectural robotic workflows still require convoluted and time-consuming programming in order to execute complex fabrication tasks. Additionally, an inability for robots to adapt to different environments further highlights concerns around the robotic manipulator as a primary construction tool.

There are four key issues currently present in robotic fabrication for architectural applications. Firstly, an inability to adapt to unknown environments; Secondly, a lack of autonomous decision making; Thirdly, an inability to locate, recognise, and then manipulate objects in the operating environment; Fourthly a lack of error detection if a motion instruction conflicts with environmental constraints.

This research begins to resolve these critical issues by seeking to integrate a feedback loop in a robotic system to improve perception, interaction and manipulation of objects in a robotic working environment. Attempts to achieve intelligence and autonomy in static robotic systems have seen limited success. Primarily, research into these issues has originated from the need to adapt existing robotic processes to architectural applications. The work of Gramazio and Kohler Research, specifically 'on-site mobile fabrication' and 'autonomous robotic stone stacking', present the current state of the art in intelligent architectural robotic systems and begin to develop solutions to the issues previously outlined. However, the limitations of Gramazio and Kohler's research, specifically around a lack of perception-controlled grasping, offers an opportunity for this research to begin developing relevant solutions to the outlined issues.

This research proposes a system where blocks, of consistent dimensions, are randomly distributed within the robotic working environment. The robot establishes the location and pose (position and orientation) of the blocks through an adaptive inclusion test. The test involves subsampling a point-cloud into a consistent grid; filtering points based on their height above the ground plane in order to establish block surfaces, and matching these surfaces to a CAD model for improved

accuracy. The resulting matched surfaces are used to determine four points which define the object rotation plane and centre point. The robot uses the centre point, and the quaternion rotation angle to execute motion and grasping instructions. The robot is instructed to repeat the perception process until the collection of all the blocks within the camera frame is complete, and a preprogrammed wall is built.

The implementation of a robotic feedback loop in this way demonstrates both the future potential and success of this research. The research begins to develop pathways through which to integrate new types of technologies such as machine learning and deep learning in order to improve the accuracy, speed and reliability of perception-controlled robotic systems through learned behaviours.

Contents

Terminology

ABB - Manufacturer of industrial robotic manipulators.

Deep Learning - The process of developing neural networks capable of generating patterns through which an artificial system can use to interact with unknown environments.

Grasshopper - A visual programming language that interfaces with Rhino through a plugin.

Linux - An open source operating system similar in function to Windows

Machine Learning - The process of allowing machines to learn the best course of action to take for a particular situation.

Operating Environment - The immediate environment in which the robot interacts with.

Pick and Place Operation - The process of a six axis industrial robotic manipulator grasping objects within the operating environment and placing these objects in order to construct various structures.

Python - A high level programming language used for a wide range of programming tasks.

Quaternion - A set of four numbers that describe the rotation of the robot around a point in 3D space

Rhino - 3D modelling software common in architectural workflows

Robot Operating System (ROS) - A framework of pre built tools used to develop complex robot control systems.

1. Introduction

The use of robots with the explicit intent of building complex architectural structures is a relatively new area of research within the field of architecture. Before 2010, the capabilities of the technology available to architects pursuing this endeavour were limited. The introduction of several new technologies and a renewed interest in developing a more efficient means of construction has once again catapulted the robotic manipulator to the forefront of construction tools in architectural research. This renewed interest has been of increased importance to the development of intelligent autonomous robotic systems designed to function in ways previously not possible, due to technological limitations.

Attempts to implement complex subsystems to control robots through artificial intelligence has seen significant interest within the last ten years. The advent of technologies such as machine learning and deep learning have presented new methodologies in which to approach these issues. Additionally, the theoretical aspects of what defines intelligence have been widely explored with an emphasis on replicating the behaviours of intelligence in artificial systems. By extrapolating intelligence into four distinct characteristics; autonomy, perception, decision making and learning, this provides a means of defining intelligence for robotic systems. All four of these characteristics, when implemented in an artificial system, develop different technical structures responsible for exchanges between the robot and the operating environment. Autonomy can be considered as the ability of the system to operate without external intervention, while perception adds a layer of complexity by preempting system actions through experience. Decision making contributes to the overall system by applying a critical lens to the information fed back from the operating environment. Learning, in contrast, extrapolates the systems previous experiences in order to formulate reasoned approaches to unknown operating environment conditions. The combination of these four theoretical markers act as a means of measuring the success of the outcomes of this research based on the importance of each aspect in defining an intelligent system.

Given these conditions of intelligence, this research develops two systems that interact with two different stages of the communication between

the robot and the operating environment. Firstly, the development of a system through which the robot is capable of gathering several blocks from within the operating environment and using these to construct a predefined wall structure at a position strictly specified by the operator. Secondly, the development of a system that consists of considerably more operator setup time however, it seeks to redefine the method through which the operator interacts with the robotic system. In this case, the operator generates a set of plans for a block wall structure which are shown to the robotic systems visual sensor and in turn interpreted into digital information which the system can use to construct the wall system. Both of these systems have opportunities for further development with the intention being to combine the two prototypes created to increase the ability of the system to build more complex architectural structures.

This research is still in the early stages of development and, while the outcomes of this research are not production-ready, they offer both a starting point and framework for the continued development of a system capable of fully autonomous construction.

2. Background

In order to begin to understand how best to implement intelligence in artificial systems, a general understanding of the underlying conceptual frameworks of human intelligence (referred to as intelligence) is required. These conceptual frameworks outline how our understanding of intelligence is defined. Moreover, significant consideration must be made to incorporate these fundamental understandings into the current discourse around the implications of implementing intelligence in previously unintelligent systems. Conceptually this framework manifests in several unique conceptual ideas, each heavily reliant on the others to develop complex, intelligent systems.

2.1 Intelligence

Consensus on how to define intelligence is not present in the literature. The lack of common understanding makes it almost impossible to define global intelligence. However, when examining the literature, common themes in intelligence exist. The commonality of these comparisons still leaves the scope of intelligence unresolved. However, several attempts have been made to offer a succinct understanding of intelligence. Binet (1905, p. 6) describes intelligence as "...adapting one's self to circumstances", similarly, Wechsler (1944) suggests the "capacity...to act purposefully...to deal effectively with the environment" constitutes intelligence. Additionally, Sternberg (1982) adds that intelligence requires goals in which to direct environment interaction.

Given these generalised conceptual viewpoints, this thesis will manifest intelligence as the ability to interact with environments and carry out adaptive, goal-orientated objectives. This definition is purposefully simplified to allow for definitive evaluation against the outcomes of this research and aligns with current approaches to implementing intelligence in artificial systems. Intelligence is the overarching concept consisting of several sub-concepts that offer more direct means in which to manifest intelligent processes for unintelligent systems.

Several other questions arise from the literature that offer additional considerations to the scope of intelligence for this research. The first, "Is

intelligence the level of competency displayed when undertaking both unknown and familiar tasks?" and secondly, "Is intelligence the rate at which a system can learn?". Answers to these questions are not directly explored in this thesis, given the complexity of the definition of intelligence. However, they are relevant to use as markers in which to reference when considering the implications of outcomes from this research.

Intelligence can be divided into six distinct subsystems that contribute to an overall definition of system intelligence. The combination of these systems provides a robotic system with the means and understanding to usefully interact with the operating environment and develop systems beyond conventional programming structures.

2.1.1 Cognition

Cognition and Intelligence are both referenced in the literature in similar terms. However, it is essential to differentiate between the two. Cognition offers additional layers of response from the intelligent system to make it more effective at manifesting intelligent actions and interactions. Key concepts that underlie the cognitive ability of intelligent systems include Knowing, Remembering, Understanding, Communicating and Learning. The most important of these for robotic systems is learning. The other four concepts rely heavily on learning in robotic systems in order to have any significant effect on the overall cognition of an artificial system.

The human ability of cognition relies on an acute understanding of two key concepts. The first, Environment Perception, provides cognitive systems with the ability to ingest information about their surroundings (Brooks, 1991). The second, goal attainment, requires the system to be able to examine the information presented to it, and in turn, respond in such a way as to succeed in its required goal (Maes, 1990; Smithers, 1997). These definitions of cognition are similar to intelligence suggesting that the two are inherently connected. Cognitive systems however, contain two additional underlying systems that operate together to define the cognitive ability of artificial systems. The first, Classification, is a means of grouping perceived environment situations into groups for easier

identification in new or unknown situations. The second, Problem Solving, uses the previously presented classifications in order to solve issues that prevent a system from achieving the required goal state. Both of these underlying systems, in combination with the previously outlined concepts of perception and goal attainment, contribute to a cognitive system. It is however, classification and problem solving, that differentiate a cognitive system from an intelligent one.

Classification, when considered in relation to cognitive systems, offers considerable comparisons to current machine learning and deep learning methods. Classification acts as a means of quickly organising the environment in distinct groupings. These groupings can then be applied to new or uncertain environments in order to quickly ascertain the constraints of an environment, what objects are present in the environment and how the system should begin to interact with the environment. Classification is not a means of accurately determining all uncertain environment parameters however, it does provide a way in which an artificial system can quickly develop approximations of the parameters and constraints to the environment it inhabits.

Problem-solving can be broken down into three categories that are typical of cognitive systems. A Trial and Error approach repeatedly modifies different parameters until achieving a suitable solution. An Algorithmic approach systematically deals with problems in a step by step process. This approach guarantees a successful outcome. The final approach, Heuristics, uses previous experience and collected information to increase the speed at which problems are solved. These concepts are all of particular importance to developing intelligent artificial systems. Cognitive problem solving draws significant comparisons to current machine learning and deep learning applications specifically through the methods that cognitive systems use to solve problems. As such the application of this knowledge will be discussed further with the specific intent on integration in current deep learning methods.

Cognition offers significant points of interest to consider what

behaviours, and abilities offer the best chance of success when integrated with artificial systems, specifically industrial robots for architectural applications. The scope of how these can be integrated is discussed further in a later section with attention paid to the comparisons with current machine learning and deep learning techniques and the potential for these techniques to successfully achieve complex robotic operations without programmed instruction.

2.1.2 Perception

Perception, the second subsystem of intelligence, has two layers of complexity that progressively offer increasingly complex manifestations of intelligence. Environment perception is a means of replicating real-world operating environments in digital form. Perception itself adds an additional layer of complexity by allowing the system to instantly prepare itself to interact with the environment, a preconceived response to ingested visual information about the operating environment and the role of objects in that environment. With the intent of determining what objects can assist the system in achieving the required goal or task. Both of these layers are vital to intelligent systems. Gathering information about the environment is essential in order to successfully interact with and manipulate objects within operating environments effectively. The added layer of an instantaneous response afforded by perception presents an increased robustness to the robot - environment feedback loop.

Commonly, perception is referred to in the literature as the ability to understand the environment through vision. However, this limits the importance of perception to the intelligence of artificial systems. This is done by reducing it to merely the gathering of information and not the reasoning behind the processing of this information that influences the reactions of the artificial system to environmental parameters and constraints. Another approach to understanding the role of perception in intelligent artificial systems is to consider the result of reacting to environmental cues presented to the system. In essence, this approach considers perception as the way

in which, primarily through vision, the system reacts and prepares itself to successfully interact with the environment it occupies.

Perception is an extremely important aspect of the autonomous robot system. Perception is not simply the act of seeing an object, it is the act of seeing a tool in the environment and adapting the system to interact with this tool. Perception is partially the way in which a system reacts to external input. When you first perceive an object you see it's uses as a tool and how you might use this to benefit your chances of achieving a certain goal or task. After this initial interaction you see an object in terms of what it is. This can be linked back to the basic need for survival, wherein the usefulness of an object in the environment is more important than what the object actually is.

2.1.2.1 Perception Limitations

The concept of perception is vital to the success of integrating intelligence, cognition and decision making in artificial systems, specifically industrial robotic manipulators. Despite this status, the difficulty of effectively implementing a comprehensive perception system capable to some extent of replicating human perception is technologically constrained. As such, perception of this type falls outside of the scope of this thesis, however, it is worth considering in future applications of this research an attempt should be made to develop an approach which is akin to human perception and in turn more effective at environment interaction and object manipulation.

For the purposes of this thesis, the role of perception is limited to simple information gathering in order to accurately replicate the robotic operating environment in digital terms. The role of reacting to this information and facilitating object manipulation is distributed among different systems. Cognitive and decision-making processes through algorithmic representations are effective at harnessing available hardware to instantiate robust intelligent robotic systems.

When considering the environment perception for applications that include robotic systems, the quality of the information gathered by these sensors is pertinent to the successful perception of different object

types. These perception systems must be able to accurately define environments in relation to the position of the environment sensors and with regard for the position of the overall robotic systems in order to offer additional functionality to an intelligent artificial system.

2.1.3 Autonomy

The use of language, such as 'autonomous' when referring to robotic systems, often confuses the significance and meaning of such terms. Smithers (1997) indicates a collision of terms from different industries that have been misconstrued when applied to newly developed robotic systems. As such, even in consultation with the literature, it is difficult to find consensus on relevant measures of autonomy through which the outcomes of this research can be evaluated.

Within the literature, attempts to define the concept of autonomy have been unable to reach a clear consensus. In contrast, some progress has been made with regards to autonomous robotic systems in specific situations. Autonomy in such systems is often described as the ability of a machine to achieve a particular task (Froese et al., 2007). Autonomy described in this way has limited applications. Pfeifer (1996) adds that autonomous agents interact with the environment without the need for human intervention. Environment interaction without human interaction is a crucial component of autonomous systems, especially those with artificial origins. Being able to interact robustly with the environment constitutes a reciprocal interaction that operates beyond pre-programmed constraints. However, Froese et al. (2007) argue that to restrict the definition of autonomous systems to this still lacks the essence of genuine autonomy; instead arguing that self-governance and decision making are fundamental when describing any autonomous system.

With regard to robotic systems, both Maes (1993) and Smithers (1997) consider a robotic system autonomous if it can extract environment information, understand the consequences this information poses and use this information to achieve specific goals or tasks. Also, consideration of the two domains of autonomy – Behavioural and

Constitutive - must be taken into consideration. This research focuses on the behavioural domain which defines certain behaviours that are typical of autonomous systems. For robotic systems, these behaviours include; collecting environment information, environment perception, localisation, decision making and motion instruction execution (Thondiyath, 2016). Currently, these behaviours are the best means of establishing and evaluating the autonomy of robotic systems. They offer direct measures in which to evaluate the outcomes of the research against and provide a direction in which to investigate the best way to begin to implement autonomy in artificial systems.

2.1.4 Decision Making

Decision Making builds upon the concepts of intelligence, cognition and autonomy presented previously. Decision Making adds additional responsibility to the system in order to manifest stronger environment interaction and better consideration for the state the operating environment is in at the time a decision is made. Additionally, decision making develops strategies to deal with issues presented in the environment that an intelligent system must overcome to be successful at achieving the required task or goal. Saaty (2008) in 'Decision making with the analytic hierarchy process' outlines the process required to be undertaken in order to make decisions: A clearly defined problem, outlined in terms specific to the situation, along with the purpose for making the decision and defining the criteria that confirm a successful outcome to the decision-making process. To manifest this in artificial systems, information about the state of the environment, what the system has to achieve and all the potential actions the system is able to take, act as inputs the decision-making process.

Within decision-making, two subcategories exist which define two typologies of the decision-making processes. The first is controlled decision making and the second is fuzzy goal or constraint decision making. Bellman and Zadeh (1970) argue that when the goal for a decision-making process is not defined, outcomes of the process are ambiguous potentially leading to undesired outcomes. Additionally, having unclear or unknown

constraints reduces the suitability of decisions made to specific outcome processes (Bellman & Zadeh, 1970; Saaty, 2008). Current technology is limited in that the accuracy of the environment model is not sufficient to preemptively make decisions about how to interact with the environment.

The integration of decision making in a robotic system can be thought of as the ability to evaluate the current state of the operating environment and, based on this information, execute instructions on the robot that makes use of this information to inform the robot's actions. This is an essential element that needs to be included in this research to allow for more complex pick and place operations to be undertaken.

2.1.5 Learning

Does an intelligent system, designed for architectural fabrication, need to learn? Should an intelligent system, designed for architectural fabrication, need to learn? These are highly relevant questions that remain unanswered in the context of intelligent robots for architectural fabrication tasks. We can store information easily in a computer, so the fact of retaining knowledge is inconsequential to machine learning. What is most pertinent, is the ability to firstly learn relationships that can then be applied to new information presented to the machine.

What constitutes learning? Is it forming relationships based on predefined inputs, or is the creation of new relationships based on a large amount of information, or are you said to have learnt something after completing it after several attempts? I would argue that human learning is a combination of all three of these characteristics and additional layers of complexity that cannot be fully understood.

Learning is not typically attempted by autonomous individuals on a daily basis. Instead Brooks (2014) argues that "Most of what people do in their day to day lives is not problem-solving or planning, but rather it is routine activity in a relatively benign, but certainly dynamic, world"(p. 1). This speaks to the vast prior experiences that human intelligence is built on. The amount of information required to achieve

similar learned results with intelligent robotic systems is virtually impossible. However, what this stresses is that given enough information through which to learn a classification based approach to the required operations, pre training algorithmic models which generalise how a robot should interact with the operating environment are a suitable solution in order to achieve simple cognitive tasks such as the picking and placing of rectilinear objects with consistent dimensions.

2.2 Feedback Loops

Feedback loops are the underlying structures present in all intelligent systems. They act as a means for systems to gather information about the environment and perpetuate this information into useful actions to further the goals of the system. Feedback Loops are the process through which information, via a number of different sensors, is gathered from the environment. Using this information the system, utilising its base intelligence, makes decisions that transition the system towards an intended goal state. Once the best course of action has been determined, the system executes the required tasks necessary in achieving the desired goal state.

This process is essential in a robotic pick and place operation due to the reliance on the robot having an understanding of the conditions of the environment it is operating in. Additionally, the robot is aware of the end goal of the system, in this case, a completed block wall structure. Both of these conditions are the fundamental principles involved in a feedback loop system. The development of a comprehensive feedback loop system for robotic pick and place operations would be a fundamental component in any application for robots in construction.

2.3 Artificial Intelligence

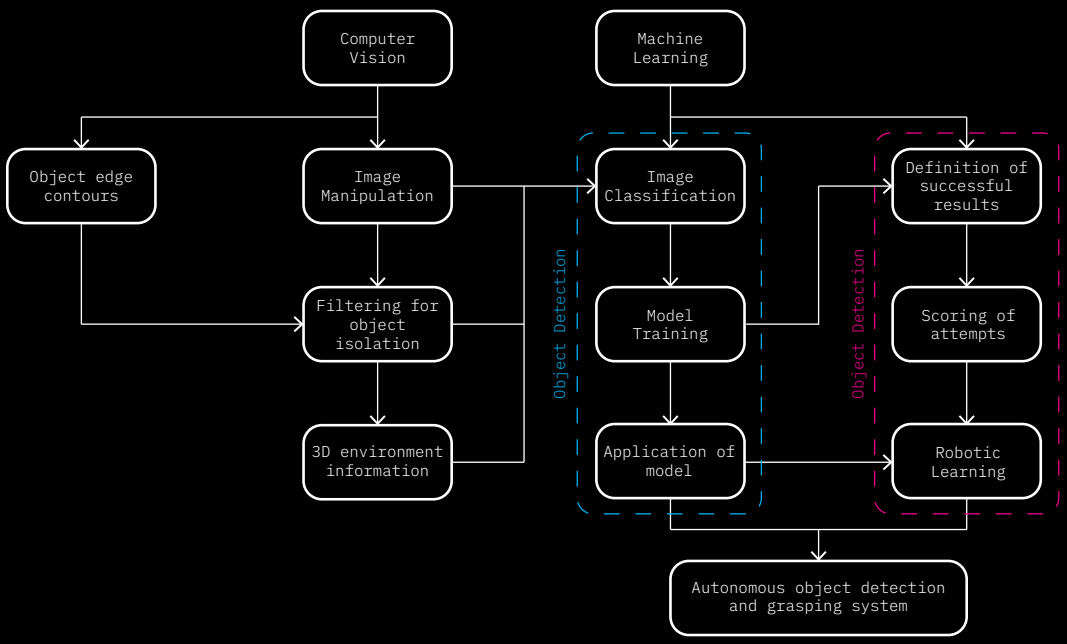
Attempts to transfer human intelligence processes to artificial systems have met limited success since they began in the mid-twentieth century. A significant number of different approaches have been considered from highly structured ‘state machines’ to more flexible deep learning approaches that have been developed in the last ten years. As such, the term ‘Artificial

Intelligence' has come to define an ever-expanding scope of concepts and processes that attempt to implement some sort of intelligence into artificial systems. Given this wide scope, it is important to clarify how this thesis defines artificial intelligence and what processes are of particular importance for the successful development of a robotic system capable of executing simple additive fabrication tasks without pre-programmed definitions.

The successful integration of simple intelligence in a robotic system will present two key concepts. The first, Environment Perception will, as previously stated, render the robotic operating environment in digital terms. The second, Decision Making will interpret the environment information, algorithmically retrieving suitable information in order to allow the robotic system to successfully achieve specific goals or outcomes. In simple terms, can the system perceive the environment and if so, is it able to understand it in such a way that it can adapt and interact with the objects in the operating environment in a meaningful way? Within this simplification of intelligence, the additional processes of cognition, autonomy, learning and feedback loops are all operating underneath the decision making process in order to manifest competent decisions in the robotic process.

Specific approaches to manifesting these intelligence processes can be split into two distinct categories. A structured rule-based intelligence, where a set of rules determine the extent that the system has to operate in. The second, is a more flexible learned intelligence, where the system has learnt formal and informal relationships about the operating environment and objects in it with which it formulates its own set of feasible considerations for interacting with the environment. Simply, rule-based vs learnt relationship. These two approaches are most common in the literature having transitioned from rule-based approaches to learning approaches with the development of more powerful computational hardware.

Machine Learning is a subset of artificial intelligence concerned with providing the necessary computational structures which would allow a machine to learn and retain knowledge (Figure 1). For the purposes of this research a subset of Machine Learning, Deep


Computer Vision
Machine Learning
Object edge contours
Image Manipulation
Image Classification
Definition of successful results
Filtering for object isolation
Model Training
Scoring of attempts
3D environment information
Application of model
Robotic Learning
Object Detection
Object Detection
Autonomous object detection and grasping system

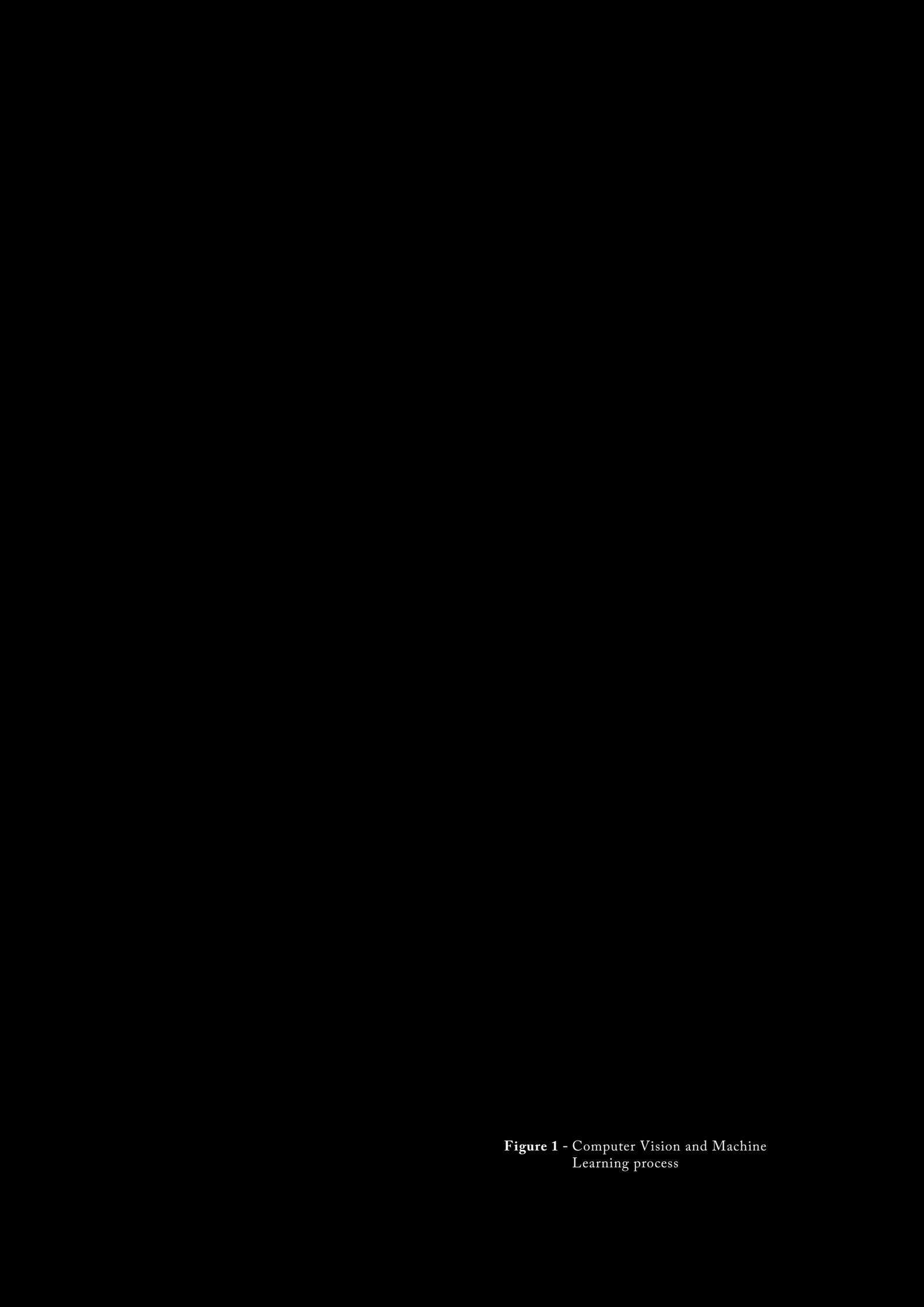


Figure 1 - Computer Vision and Machine Learning process

Learning, is found to offer the appropriate scope to successfully develop autonomous robotic pick and place operations.

2.3.1 Deep Learning

Deep Learning is a recent attempt, having re-emerged as a viable approach in the last ten years, to develop robust and effective intelligence implementations in artificial systems. Deep Learning formulates algorithmic models that can be applied to new situations consisting of similar data to facilitate advanced levels of intelligence in specific applications. Deep Learning training methods can be defined in three distinct categories; supervised learning, unsupervised learning and reinforcement learning. These three categories attempt to achieve learning through different approaches, the advantages and disadvantages of which will be discussed subsequently. Each approach provides a different means of achieving a level of intelligence in artificial systems.

Supervised learning attempts to train a model, representative of a predefined relationship in a data structure. Each data point consists of a description and a label which defines what action should be taken (Sutton & Barto, 2018). The purpose of this system is to develop generalised relationships that allow the system to operate effectively in unfamiliar or uncertain situations. In addition, supervised learning applies basic rules upon which the system can define the representative model (Guérin et al., 2018). The combination of these two control processes define a model which is adept at a singular trained task.

Supervised learning presents both clear benefits and limitations. For the purposes of interacting with the environment, the collection of correct action representations is often impossible (Sutton & Barto, 2018). Additionally, supervised learning is unable to adapt to any change to the situation in which it was trained. As a result, changing any of the environment parameters renders a model trained in this way unable to operate successfully, or the outputs and actions of the system will be ineffective or unintended in relation to the outcomes expected by the system designer. Given these constraints, a supervised learning model

is ineffective when deployed in a situation that requires the artificial system to be flexible and adaptable to a variety of operational situations. Therefore this learning method has been discounted in favour of methods more suitable to solving the previously presented issues.

Unsupervised learning develops representational models that, in contrast to supervised learning, determine the relationship through the model rather than being predefined (Goodfellow et al., 2016). The unsupervised learning model draws its own comparisons from the input data and as such extracts feature sets potentially unseen by the system designer (Guérin et al., 2018). The overall goal of unsupervised learning is to classify data into different feature sets in order to be applied to, in a similar fashion to supervised learning, unfamiliar or uncertain situations.

Unsupervised learning has several benefits when compared to supervised learning but is limited as a means of establishing flexible and adaptive robotic grasping systems. The benefits of unsupervised learning are present in the ability for complex situations to be accurately represented in the learnt model. The relationships with the training dataset also have the potential to be classified in ways that are not immediately evident but still produce the desired results and outcomes. In contrast, the limitations of unsupervised learning restrict the adaptability of the model when the system is introduced to situations in which the training data did not consider. The unsupervised learning model offers some benefits over the supervised model however the limitations still indicate that it lacks the means to develop flexible and adaptable robotic systems for architectural fabrication situations.

Reinforcement learning is the process of interacting with the environment and learning in real-time. Specifically, reinforcement learning focuses on operating environment goal states which the system interacts with, in order to achieve a goal or task (Sutton & Barto, 2018). For each action the system takes towards achieving a goal state, it is given a score or reward which indicates how effective that particular action is in achieving the goal state. Given the importance of the environment state, it is essential that the system is able to accurately reflect changes made to the environment in a way that

the reinforcement learning system can utilise to inform the next action.

In contrast to both supervised and unsupervised learning, reinforcement learning is capable of operating in real time. This gives it the distinct advantage of being able to adapt the learning process as the system encounters different operating environment situations. However, to ensure that a reinforcement learning system acts effectively, consideration needs to be given to the trade off between exploitation and exploration. The system must exploit its current knowledge to guide actions but it must also explore different actions in case a new action is more effective than a current action (Sutton & Barto, 2018). Reinforcement learning offers the best solutions to the complex issue of flexible and adaptive robotic systems for additive architectural fabrication, consistently meeting the requirements as outlined previously.

Given the goals of this research, reinforcement learning has been used to begin to implement complex intelligent decision making in a robotic system. The benefit of real time operation significantly outweighs any of the limitations and allows the robotic system to be flexible and adaptable to different situations.

2.3.1.1 Object Detection

Deep Learning begins to facilitate more complex intelligence processes, the most basic of these being object detection. Object detection refers to the ability of an artificial system to detect when objects are within the perception frame. 2D object detection is common where a bounding box is applied to the region where the deep learning algorithm believes the object is. Common applications for object detection systems are in autonomous vehicles, industrial and domestic robots. This research uses those systems as a baseline for the integration of object detection in additive robotic fabrication for architectural applications.

Object detection can be developed using three different approaches. Sobti et al. (2018) approach object detection using simple RGB images in which a rectangular bounding box is applied when a deep learning model

successfully detects the specific object in the frame. Rahman et al. (2019) add an additional layer of complexity by including depth information in the form of RGB-D images. The added depth data is an attempt to improve the 2D detection process to include 3D environment information in the object detection process. Ziegler et al. (2018) go one step further to render an RGB-D image as a point cloud. The point cloud is matched to a CAD model of the object to be detected with the intent to improve the accuracy of the object detection process allowing for better robotic - object interaction.

2.3.1.2 Object Recognition

Object recognition builds on the process developed in an object detection system to allow an artificial system to know what objects it is perceiving through an environment sensor. Object recognition is linked heavily with the concept of perception. The purpose of object recognition is to establish the usefulness of objects in the operating environment, quickly establishing through a pretrained model, how best to use objects presented to the robotic system in achieving a set of goals or tasks. This adds an additional layer of sophistication to an intelligent system over simply identifying what objects are present in the perception frame.

The scope of approaches to object recognition can be demonstrated in three studies. Baareh et al. (2012) use feature extraction through an artificial neural network to develop a robust method for 3D object recognition. Pinto et al. (2013) integrate a laser range finder into the object recognition system in order to overcome some of the limitations of camera based recognition systems, specifically the requirement for suitable lighting to evenly light the object to be recognised. Nagy (1994) attempts to use pattern recognition to differentiate between different object types during the recognition process. Of these three approaches the most useful is Barreh et al. as a system that requires robots to interact with objects necessitates that object recognition is highly accurate to reduce errors in the object grasping process.

2.4 Robotic Systems

Robots, specifically those suited to industrial applications, have long been

used as a means of increasing efficiency and reliability in a number of industries. Currently, industries that benefit most from these productivity increases are those in which robots are required to undertake the same task repeatedly e.g. automotive manufacturing. Contrasting this, the introduction of robots as tools in industries that require more adaptability and flexibility from robot systems has been limited. Architectural applications, specifically for construction, require robotic systems that function reliably across a wide range of scenarios and conditions from prefabrication in a controlled factory environment to on-site construction in a complex and challenging site. These limitations have long restricted robotic use in architecture to trivial applications. However, with the advent of new technologies coinciding with the 4th industrial revolution, robots are beginning to gain interest as feasible construction tools for both prefabricated and on-site construction.

Robots have been used since the mid-twentieth century as tools to increase productivity in specific industries. Robots are inherently good at accurately repeating the same task. However, for applications that require flexibility and adaptability in robotic motion instructions with specific consideration for environment interaction, current robotic processes cannot fulfil these demands. Architecture, and specifically the construction of architecture, is one such industry where flexibility, speed, efficiency and adaptability is paramount. As such, the current software tools dictating robot motion control are fundamentally unable to manifest complex architectural structures from robotic assembly processes. Furthermore, the disconnect between a robot and its operating environment underlines the limitations of current robotic systems to be utilised as construction tools for architecture applications.

Attempts to introduce robots as tools for the construction of architecture is evident since the widespread uptake of robots as manufacturing tools in conjunction with the advances in software control for the programming of robots. Since this time, attempts to integrate robots into robust construction workflows has significantly increased. In addition to this increase, significant attempts have been made to introduce tools that allow the robot to perceive and interact with the environment that it

is operating in. Primarily these systems have been developed through the work of the Gramazio and Kohler research group. Their research focuses on integrating robots into construction workflows with increased speed, efficiency and flexibility over typical construction workflows.

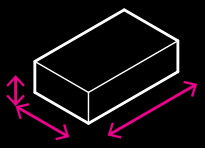
2.4.1 Intelligent Robotic Systems

Applications for computer vision and autonomous robotic decision making are vast. As a result of this, a significant amount of research already exists that considers how to implement computer vision in an industrial robotic workflow. The majority of the research deals with mobile robots with the intention of having robots navigate through an environment while only a very small amount of research deals directly with the application of robotic feedback loops for architecture and construction applications. The existing research can be broken down into two main categories. Firstly, object mapping for 3D model generation and secondly, object detection and recognition for industrial robotic handling

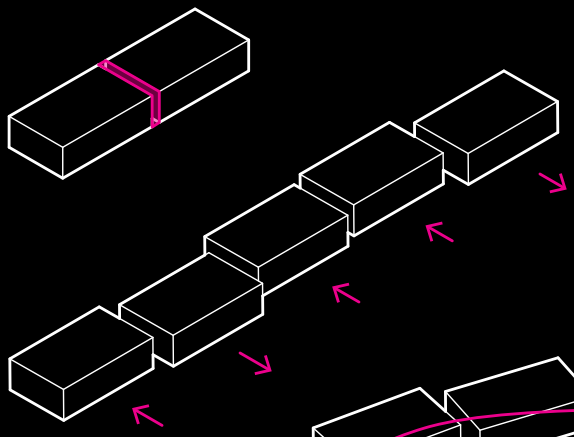
Preliminary work on 3D object mapping was undertaken by Besl and Mckay (1992) and Blais and Levine (1995). This preliminary work attempted to use rudimentary vision systems in order to generate digital 3D models. Similarly, Newcombe et al. (2011) in the paper KinectFusion: Real-Time Dense Surface Mapping and Tracking, presents an updated implementation of vision sensing and object recognition. An RGB-D sensor captures both image and depth data, on which the authors implement an iterative closest point algorithm in order to combine depth data from different viewpoints into a descriptive and complete digital model. More recent research, undertaken by Martin et al. (2014), investigated the implications of interference on the reliability of depth measurement data collected from RGB-D sensors. This research highlighted the need for adaptive algorithms to account for image interference.

Research that considers the implications of intelligent and environmentally aware grasping in an industrial robotic workflow is limited to a few key industries. As such, intelligent grasping approached from an architectural perspective, has seen limited research interest. To date, two key studies

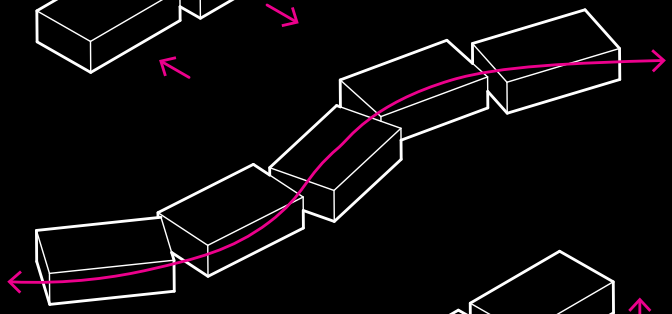
1.



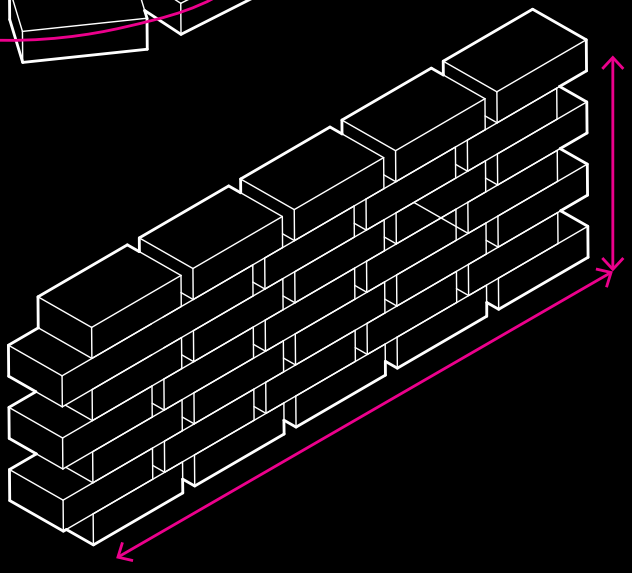
2.



3.



4.



5.

Figure 2 - Block wall Parameters

1. Block Dimensions.
2. Mortar spacing.
3. Wall offset.
4. Wall path
5. Overall wall dimensions

have investigated the integration of object detection in a robotic workflow. De Gregorio et al. (2016) in the paper "RobotFusion: Grasping with a Robotic Manipulator via Multi-view Reconstruction", describe a means of using a robot equipped with an RGB-D sensor to determine the location of objects in an environment in relation to the robot coordinate base. A second investigation by Tsarouchi et al. (2016) examined an offline system that utilises CAD files of objects to match to scanned objects. More specifically, the pose of the robot is determined through similarities and differences in the RGB images gathered by the robot. One further example by Furrer et al. (2017) in the paper "Autonomous robotic stone stacking with online next best object target pose planning" presents the current state of the art with regard to object detection in a robotic workflow in the architectural and construction industries.

2.5 Robotic Brick Construction

The use of robots in architectural applications has increased significantly since the beginning of the 21st century. This uptake in robotic use has also coincided with the introduction of additive fabrication processes for robotic construction. Primarily, this has taken the form of six-axis industrial robots placing bricks in order to construct bespoke wall systems (Figure 2).

Some research exists that examines the implications of the introduction of robots in simple additive fabrication processes. Two systems developed in the 1990s were the first attempt to implement robots in adaptive construction processes. However, these systems were primarily focused on improving efficiency without considering the potential that new fabrication technologies could have on the ability to construct significantly more complex wall geometries. As such, they failed to gain any traction within the construction industry.

These systems all have significant limitations that revolve around a lack of operating environment awareness, an inability to adapt to new operating environments and the inability to adapt to changes in the existing operating environment. These limitations severely restricted the suitability of these early systems to successfully construct

complex block wall systems with limited human intervention. However, these initial attempts developed a framework on which additional development and research could be undertaken.

2.5.1 Intelligent Robotic Brick Construction

The first of these precedent examples titled 'Flexbrick' aimed to develop a means of prefabricating sections of brick walls using an industrial robotic manipulator (FlexBrick, ETH Zurich, 2008-2010, n.d.). In contrast to the proposed research, this project did not use any adaptive or autonomous feedback systems to enhance the process of fabrication. This project is a useful marker in terms of demonstrating the viability of construction that utilises bricks and the potential for robotic fabrication to begin to develop systems that utilise brick construction. The proposed research has the potential to build upon this project in an attempt to introduce flexibility and adaptability into simple architectural robotic fabrication tasks.

The second precedent example is again a project undertaken by Gramazio and Kohler Research. In contrast to the previously outlined precedent, this project begins to incorporate sensor data in an autonomous feedback loop. The project utilises this data to determine the width of timber panels that are used to clad premade structural timber frames. This process is rudimentary in sensor application as the timber cladding is placed in the same position for every operation and therefore cannot be adapted easily if the environment the robot was operating in was to change (Eversmann, 2018).

The third precedent project titled 'Stratifications' is a project that uses vision-based feedback in order to determine what thickness of brick should be picked in order to maintain the structural integrity of a block wall system (Figure 3). This project uses an RGB-D sensor that scans the already constructed wall in an attempt to determine the dimensions of the blocks surrounding where the new block is going to be placed. Based on these measurements, the algorithm decides which block is best suited to fill the gap, and the robot then executes a manipulation instruction in order to grab the block from one of three

Stratifications

The Endless Wall

On-site Mobile Robotic Construction

Figure 3 - Stratifications.
(2011).
Gramazio and Kohler Research.
https://gramaziokohler.arch.ethz.ch/web/e/projekte/206.html

Figure 4 - The Endless Wall.
(2011).
Gramazio and Kohler Research.
https://gramaziokohler.arch.ethz.ch/web/e/projekte/216.html

Figure 5 - Building Strategies for On-site Robotic Construction.
(2014 - 2018).
Gramazio and Kohler Research.
https://gramaziokohler.arch.ethz.ch/web/e/forschung/273.html

different stacks. These stacks of blocks are in known locations, and this is where the proposed research differs significantly. The proposed research aims to replace known location object picking with vision-based object picking in order to reduce setup time and remove errors from the picking operation (Stratifications, London, 2011, n.d.).

The current state of the art with regard to autonomous and intelligent robotic construction methods is present in the in-situ fabricator (Helm et al., 2014) (Figure 5). This is a mobile robot with the ability to operate in a range of different environments. Specifically, this mobile robotic system is capable of construction in two distinct additive fabrication processes. The robotic system is capable of constructing pre programmed brick wall structures in an area larger than currently possible with fixed based robotic manipulators. Additionally, the robotic system is also capable of fabricating complex steel mesh reinforcing structures currently uneconomical with traditional construction processes. These two additive fabrication systems are achieved through the use of vision sensor information and complex algorithmic systems which provide the robotic system with the information required to actively construct these structures. As the robot is also required to move around the environment, it uses the collected environment information to generate a rudimentary map of the operating environment, which is updated every time the robot moves. This allows the robot to localise itself within the environment and understand its position in relation to the fabrication process it is executing. While this system is significantly complex, it still has several limitations which impact the autonomy and flexibility of the system to undertake certain tasks. Firstly, the system requires human input to transfer material to the robot. In addition it is not able to grasp objects from locations that are not preprogrammed into the system. The system is also untested in environments that are not highly controlled and as such is not capable of sufficient autonomy outside of a laboratory environment.

2.6 Autonomous Vehicles

Over the last ten years, the success of autonomous vehicle in real-world

applications has increased significantly. The development of complex systems, which allow vehicles to operate without human intervention in complex environments, offers a framework or roadmap for the integration of similar systems in the architecture and construction industries. This roadmap consists of developing processes to control the vehicle with computers, including implementing several different sensor arrays to accurately map the operating environment. Further processes include developing intelligent algorithms to make sense of the information gathered from the sensor arrays and then developing these algorithms by implementing machine learning technology into the system to improve the overall capabilities of the system by learning how a human would operate a vehicle.

This roadmap can be applied to a robotic pick and place operation as it will require the implementation of similar systems in order to function successfully. Firstly, the robot will need to be controlled using external applications and in real-time. Secondly, in order for the robot to understand the extent of the task, it is required to fulfil, a minimum of one visual sensor will be required. Thirdly, the robotic system will require the implementation of intelligent algorithms in order to successfully use the information available to execute autonomous pick and place operations. The similarities between autonomous vehicles and autonomous robots for construction provides a useful framework for the development of a robotic system with a comparable level of intelligence.

3. Aims and Objectives

3.1 Aims

Based on the previously outlined research, the aims of this research are clear in terms of both scope and position within the overall body of research. The existing research offers four critical areas in which this research begins to develop solutions. These relate specifically around developing systems that allow robots to begin to formulate intelligent decisions about what actions to take within an operating environment to best achieve the required goals of the system. Robotic feedback loops present a significant opportunity to develop new systems for more efficient and reliable construction due to the limited flexibility and autonomy of robots in current construction applications. This research aims to begin to resolve four critical issues for robotic fabrication in architecture and construction:

- The inability to adapt to unknown environments (Figure 6 - 9) .
- The lack of autonomous decision making in robotic workflows.
- The inability to locate, recognise and then manipulate objects in the working environment.
- The lack of error detection if a motion command fails to execute or is executed incorrectly.

The proposed research aims to begin to solve these issues by developing a complex autonomous robotic feedback loop system, capable of constructing a block wall using objects found through vision-based feedback. The aim is to integrate RGB-D and tracking camera environment information with object detection and recognition algorithms in order to identify the location of objects in the robot's environment, determine what the object is and also the pose of the objects in relation to the robot coordinate base. Furthermore, the successful integration of these systems within this research will begin to facilitate intelligence in architectural robotic fabrication.

3.2 Scope

The scope of this thesis is directed towards designing a successful autonomous robotic pick and place operation. The process through which

this thesis will operate interrogates the literature for a means to define the extent of the research scope. The scope of this thesis investigates four key areas of the literature and their applications to robotic pick and place systems- Intelligence, Autonomy, Decision Making and Perception. These four areas scope the relevant literature, ensuring that the thesis has explored the relevant areas of both theoretical and practical applications.

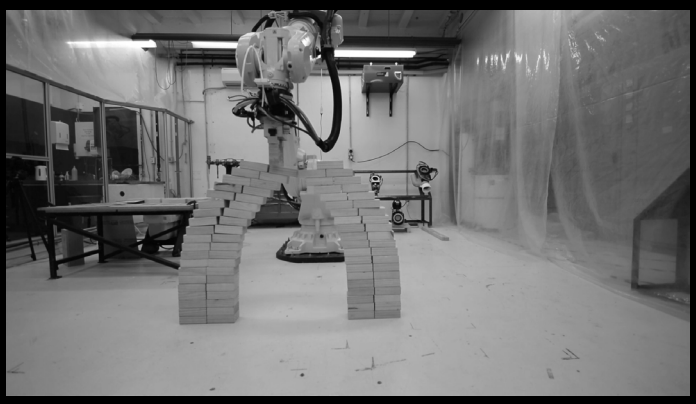
The scope of this thesis is framed in such a way as to ensure the relevance of the resulting outcomes to the overall body of literature inclusive of both theoretical and practical applications. Specifically, practical applications investigated the implications of the technical systems that control the robot, how the robot interacts with the operating environment(primarily through vision),and the complex systems that decipher the information provided by the robot interacting with the environment. In relation to the theoretical aspects of this thesis, specific conceptual ideas, as discussed earlier, are matched with the practical aspects to form a cohesive vision for the thesis.

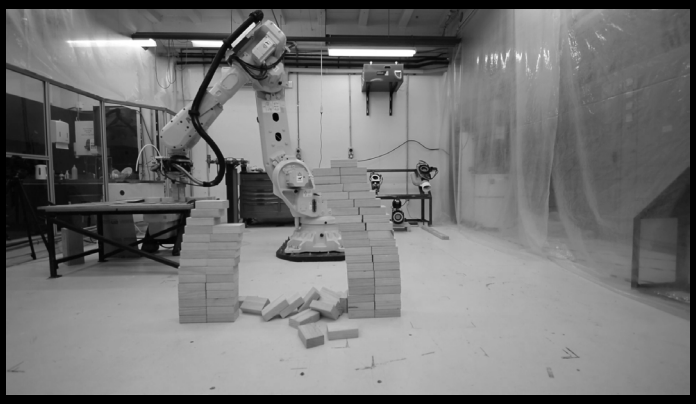
3.3 Objectives

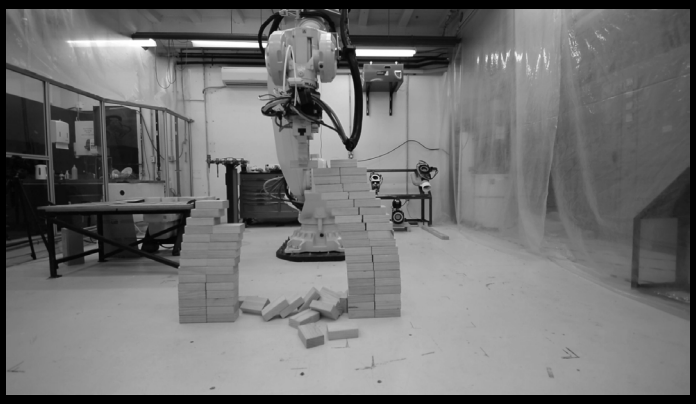
This thesis consists of several objectives in order to successfully achieve the overall aim of the thesis in developing an autonomous robotic system capable of grasping unknown objects in the working environment. The first of these objectives consisted of investigating the hardware and software tools required in order to implement feedback loops in a robotic workflow.

3.3.1 Prototype One

- Connect a simulated robot to a robot
- Have the robot respond to commands in real-time
- Gather a point cloud representation of the operating environment
- Isolate the point cloud information that describes the location blocks to interact with
- Convert the point cloud description to useable information for the robot







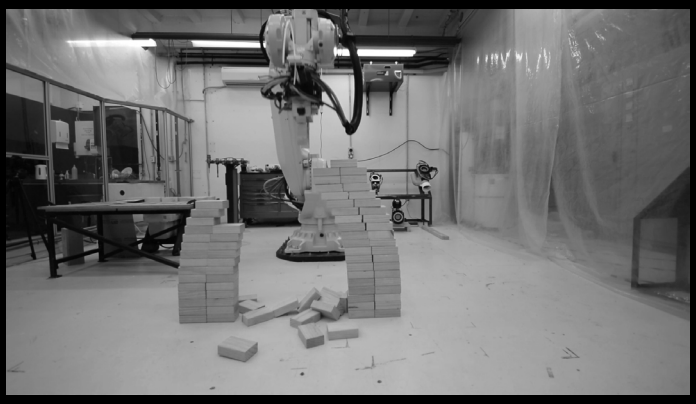


Figure 6 - Conventional Pick and place operation

Figure 7 - Operating environment does not match the programmed parameters of the robotic system

Figure 8 - Robot continues executing pre-programmed motion instructions

Figure 9 - Robot is unaware of multiple failures. Requires human intervention to correct the robots actions.

- Execute a single grasping operation

3.3.2 Prototype Two

- Loop over the execution instructions
- Implement a closest point algorithm in order to determine corner points of each block

3.3.3 Prototype Three

- Develop a random wall pattern generator
- Have the robot recognise the wall pattern from an image
- From a known stack of blocks, have the robot construct the wall from the image

3.4 Methodology

This research developed through a methodology common in both architecture and design. The design thinking methodology consists of four key stages, each reliant on the previous, to guide the direction of the research (Figure 10). In addition to this reliance is the feedback process between each stage of the design process whereby outcomes discerned in the later stages of the research are used again as inputs in the design process to influence the iterative process of the research.

The first stage in this methodology is to clearly and adequately define the problems and issues this research is attempting to address. The issues that are to be addressed revolve around the current limitations in autonomous robotic fabrication in conjunction with the formalisation of tools and processes suitable to begin to implement autonomous robotic control of the construction process. In order to materialise the extent of the research, the issues and problems need to be suitably defined and considered in order to provide a clear direction for the research to take. In addition to the initial scoping of the research, as the research progresses, the outcomes of the research also shape the scope through the iterative

design process which in turn allows the research to progress in ways which may not have been considered at the beginning of the research process.

The next stage of the research methodology is to determine ways in which the previously defined issues and problems could be solved. In the case of this research, the main ways in which ideas were generated related to solving the technical complexities of having a robot operate autonomously and begin to understand the environment it is operating in. The ideate stage of the research methodology relies on the resolution of the define stage to be at a suitable level in order to manifest solutions that have the highest potential to be successful.

The third stage in the design thinking research methodology is the prototyping phase. In the context of this research, the prototyping phase consists of three different prototypes with increasing levels of complexity, both in terms of technical development and robot action in the environment. The first of the three prototypes are developed solely on the approaches developed in the ideate phase. However, the additional two prototypes build on the previous prototypes in addition to the approaches developed in the ideate phase. After the completion of each prototype, the prototype is evaluated through a the testing phase in order to then determine where improvements can be made to the next prototype.

The final stage of this methodology is the testing phase. This phase consists of the outcomes of the prototypes being evaluated against the effectiveness of solving the previously defined problems. In the case of this research, each prototype was considered based on how effective it was on three different metrics. The first of these metrics, autonomy, considered how much human intervention and setup was required in order for the robot to successfully complete the required task. The second metric, intelligence, considers how suitable the robot's actions are when compared to how a human would approach a situation. The third metric, reliability, considers how many times the robot could successfully complete the requirements of the prototype, given different operating environment conditions. The combination of these three metrics were used to determine what aspects

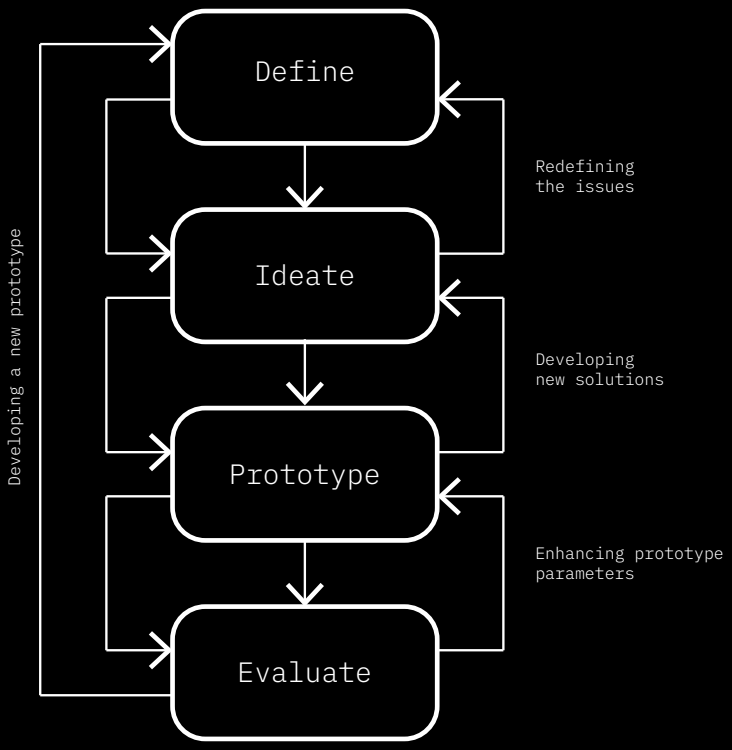

Define
Ideate
Prototype
Evaluate
Developing a new prototype
Redefining
the issues
Developing
new solutions
Enhancing prototype
parameters

Figure 10 - Research Methodology

of each prototype were successful and could be carried over to the next prototype. The metrics also identified areas where changes were needed for the next prototype in order for the outcome to be more successful.

The overall research methodology will consist of four unique phases that, when combined, form a comprehensive means of undertaking architectural research. Most notably, this methodology is useful as a way of quickly iterating through design options in order to develop robust outcomes. The use of prototyping in an autonomous robotic feedback loop system provides tangible feedback as to the effectiveness of each iteration.

The previously outlined methodology is particularly useful in developing outcomes that are highly resolved. Lucas(2016), in his book “Research Methods for Architecture”, describes the benefits of an iterative process “The results of an experiment can be unpredictable, but this is actually a primary benefit of the process – it allows you to design your next experiment to ask a more specific question”. The proposed research method allows for incremental improvements in the competence of autonomous robotic feedback loops to be observed. These incremental observations are vital in establishing robust research outcomes. They allow for the initial research question to be thoroughly interrogated thus establishing a means of determining the viability of autonomous robotic feedback loops for architectural fabrication applications.

4. Tools

4.1 Rhino and Grasshopper

Grasshopper and Rhino are Windows based systems that provide a means to simplify the process of interfacing with complex robotic systems. Rhino is a 3D modelling software package that is a standard application in architectural practice, along with many other industries. Grasshopper is a plugin for Rhino that offers two distinct new capabilities. The first is the ability to construct complex parametric geometry in the Rhino environment and secondly, Grasshopper allows the import and export of data in Rhino via additional scripting languages.

Information integration between different tools in a feedback system is essential to a successful robot operation. Grasshopper facilitates this through the integration of Python scripts that translate grasshopper geometry into readable instructions for robotic execution. Additionally, Grasshopper acts through a simple text file interface, sending files over a network, to communicate with the various other tools such as ROS (Robot Operating System), an open source,Linux based application that facilitates a live interface between generated data and robotic operation.

Using Rhino and Grasshopper as an interface between complex robotic systems can provide three distinct advantages. Firstly, for architectural applications, Grasshopper is a familiar tool which can make it easier to understand and visualise complex robotic concepts. Secondly, through visual representation, Grasshopper allows for an understanding of complex spatial information about the underlying feedback process. Finally, Grasshopper offers a familiar language interface between the robotic control structures and communicated feedback information.

While the benefits of using Rhino and Grasshopper as an interface to ROS are significant, several limitations are also apparent that restrict the usefulness of this process. Firstly, new processes on top of the already complex robotic control structures lead to potential reductions in the speed and efficiency of the overall feedback loop system, especially when operating in real-time. Secondly, the robot operating between two environments, Linux and Windows , reduces

the flexibility to adapt developed systems to different situations.

The use of Rhino and Grasshopper requires an awareness of these limitations and working to ensure they don't have a detrimental impact on the robot operating system.

4.2 Robot Operating System

The Robot Operating System (ROS) operates on a Linux based operating system and is a control structure capable of managing a wide range of robotic systems, including six axis robotic manipulators. Control of these systems is facilitated through five subsystems; Nodes, Topics, Messages, Publishers and Subscribers. The combination of these subsystems, through a communication hierarchy, establishes control over environment interaction and instruction execution in a robotic system. The communication hierarchy also allows ROS to facilitate both delayed and real-time feedback loops. This allowance for real-time feedback loops presents ROS as a framework in which to control robotic motion through environment interaction.

A series of interlinked Nodes control the interaction between ROS and a robot (Figure 11). Each node in the system handles one area of robotic control. Three distinct Node clusters are responsible for: robot motion control, external information input and, programmed information input. Robot motion control nodes collaborate to execute instructions on the robot. External information input nodes gather information about the operating environment of the robot. Programmed input nodes allow the programmer to define information about the environment or objects in the environment.

The ROS Node communication structure allows for the transfer of information about the operating environment to the robot control systems. Facilitating environment information transfer requires exploiting an array of environmental sensors. Systems such as Intel's Realsense D435 allows the analogue environment to be accurately established in digital form. Communication between the environment sensor array and the robot control structures is facilitated through ROS Topics. These Topics publish information that additional adaptive algorithms can manipulate into robot

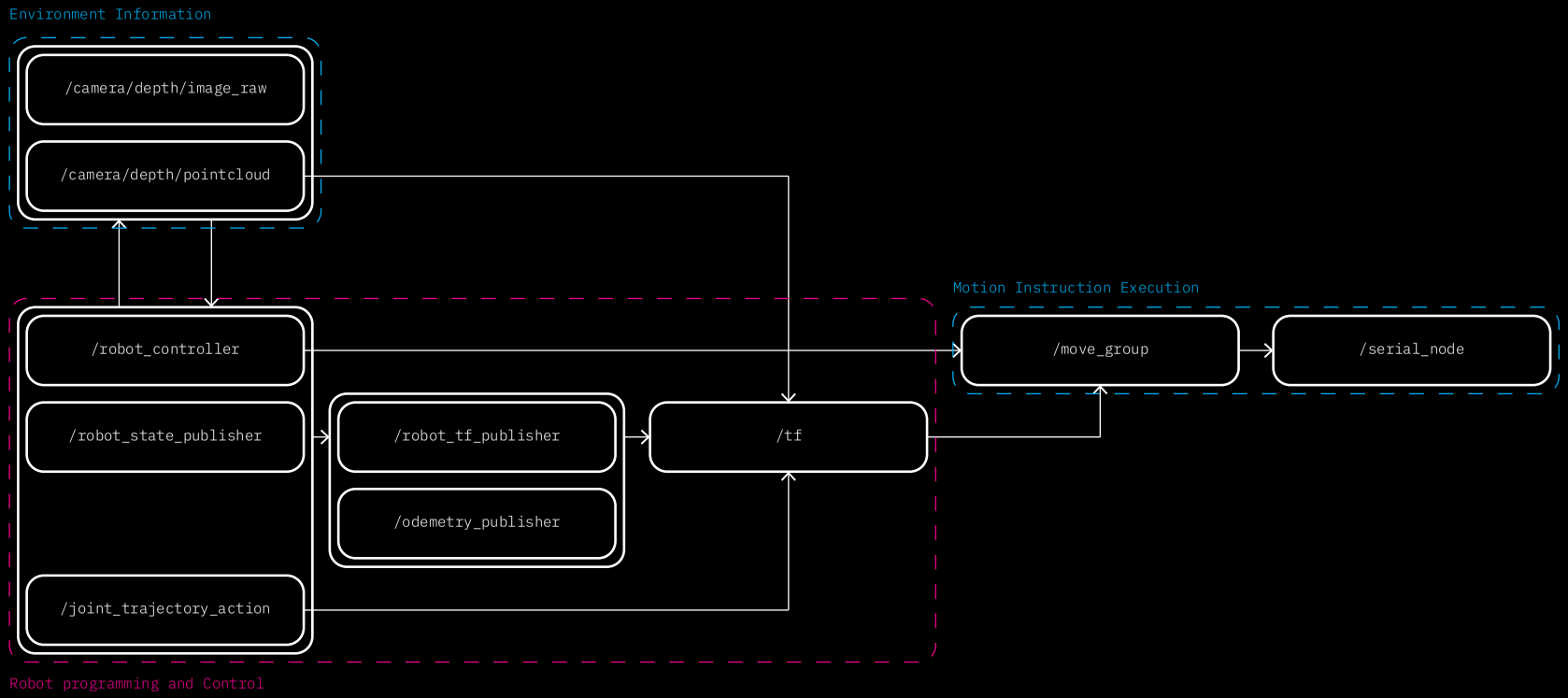

Environment Information
/camera/depth/image_raw
/camera/depth/pointcloud
Motion Instruction Execution
/move_group
/serial_node
/robot_controller
/robot_state_publisher
/robot_tf_publisher
/tf
/odemetry_publisher
/joint_trajectory_action
Robot programming and Control

Figure 11 - ROS node and topic structure

motion instructions, which are communicated to the robot and executed.

ROS Topics act as the primary subsystem for node communication. Topics act as start and endpoints for data communication. Nodes contain either a collection of Topics or a single Topic through which information is transferred in the ROS message format. To send and receive information, Topics publish and subscribe to Topics grouped in other nodes. By publishing information, other Topics can receive this information by subscribing to the publishing Topic. Topics within the same node also communicate in this manner. The functionality of Topics defines a flexible information transfer process, capable of seamless real-time feedback.

This programmed information input is essential in allowing interaction between industrial robots and operating environments. ROS facilitates the integration of programmed inputs through the generation of ROS Topics publishing such information. However, these programmed inputs are not static. Through the publishing and subscribing system, other ROS nodes can retrieve programmed data. In static robot control structures this process is mono-directional, whereas ROS facilitates bi-directional feedback which interacts with the programmed input to manifest environment parameters in robotic actions. This feedback process presents robot operators with the central systems to implement simple reciprocity between the robot and its operating environment.

In order to entertain real-time feedback, ROS communication operates in a hierarchical structure. This hierarchical structure consists of two communication layers, with a diminishing level of priority. The highest layer, i.e. the one with the highest priority, controls robot motion instruction execution. Also, other essential communication, crucial to the successful operation of the robot, is facilitated in this layer. The sublayer primarily controls sensor integration, programmed environment parameters and functions which are not critical to the successful functioning of the robot. Given that these two layers exist independently, real-time feedback between the environment and the robot control structure, along with feedback about the robot's position and state are simultaneously

updated to inform subsequent motion instruction execution.

These Feedback loops, in conjunction with ROS, suggest an approach to two-way communication for robotic pick and place operations. ROS facilitated two-way communication that incorporates environment information will result in a robotic system that is influenced by and responsive to the operating environment. Two distinct types of feedback are apparent when examining the capabilities of ROS communication. These pertain primarily to real-time and delayed feedback. Real-time feedback loops, as the name suggests, allow the robot to receive information about the environment through ROS as it is interacting with it. In contrast, delayed feedback loops update motion control systems after an amount of time has elapsed. Real-time feedback implemented through ROS has several advantages most notably, robots interacting with operating environments in real-time can adapt quickly and effectively to both subtle and significant changes in operating environment conditions. Contrasting this, delayed feedback loops require the robot to suspend the execution of instructions until environment information can be generated, and therefore allow the robot to interact successfully with the operating environment. This limitation suggests a real-time feedback loop implementation, through ROS, as the most effective means of developing two-way communication for robotic pick and place operations.

In collaboration with real-time feedback loops, ROS allows for the transfer and filtering of specific environment information. In addition to simple information communication, ROS begins to conduct simple decision-making concerning the execution of robot motion. Existing robot control processes, as outlined previously, establish simple code to robot transfers as a means of instructing robot motion. However, through the feedback loop process, ROS can establish control over numerous robotic systems. The output of each of these systems is used by ROS to determine how best to interact with the information available to it and rudimentary decision making can be established.

4.3 Intel Realsense Camera

The Intel Realsense camera (D435) contains an array of environment information gathering sensors used to generate accurate descriptions of environments in digital form. The D435 consists of a standard RGB camera module, Stereo depth cameras and an IR pattern projector. Combined, these three sensors are capable of accurately mapping the environment between a range of .1 and 10 metres. This sensor array can accurately generate several output types which depict the physical environment. The first of these is an RGB image, which is consistent with images typically captured with an ordinary camera. The second is a pixel mapped depth image, which depicts what the camera can see with the colour of each pixel indicating the distance from the camera. The third is a point cloud, which describes the environment in three-dimensional point coordinates. These three information types lend themselves to different approaches for feedback loop implementation in robotic workflows by way of the environmental representation they generate.

4.3.1 Environment Representation

In order to utilise the information provided by the environment sensors, two distinct outputs are available to allow the robot to begin to interact with the environment. Stereo depth images provide the robotic system with information that pertains to the structure of the operating environment. However, this information is not localised in regards to the position of the camera on the robot and the position of the robot in the operating environment. In contrast, point cloud descriptions provide a localised environment map that is accurately linked to the position of the camera and in turn, the position of the robot. Given the constraints of the stereo depth images, maximising the use of a point cloud in this system is the most effective means of providing an appropriate environmental representation in a robotic pick and place operation.

4.4 Tool Summary

The combination of the outlined tools presents a systematic strategy on which a comprehensive two-way communication approach to

robotic environment interaction can be established. Rhino and Grasshopper, standard architectural tools, offer a means of interfacing with complex robotic systems. This interface is perpetuated through the ROS communication system allowing for complex feedback and rudimentary decision making. Furthermore, the introduction of Intel's Realsense camera and point cloud descriptions, provide the necessary environment information in order to evolve a sophisticated robot and environment communication structure. All these tools integrated together have the potential to provide a means of formulating complex robotic operations for architecture-specific tasks.

5. Preliminary Investigations

5.1 Connecting to the robot

The connection between the robot and a control source, e.g. ROS was essential to the success of this research. This connection was facilitated through several pre-existing functions included in the ROS library. The first of these functions uses communication over a local area network (LAN) to gather information about the robot state for the control system to use. The second function uses this same LAN in order to update the state information for both the robots kinematic functions as well as the tool attached to the robot. The combination of these two functions allows for a connection to be manifested between the robot and external robot control sources.

Using this method of connecting to the robot offers two primary benefits; instantaneous communication and flexibility in the control structure. The capabilities of the control system did not restrict the scope of the research. Through implementing the connection between the robot and the control source, ROS has been determined as the most appropriate tool due to the number of different control functions available. These control functions, as outlined in the previous chapter, differentiate ROS from the other tools available. Using this control source has allowed the scope of the research to develop further than initially considered due to the complexity of the systems available within the ROS framework.

5.2 Initial feedback loop

Several different approaches were considered in order to establish essential feedback between the robot and the control source. The first uses an isolated ROS environment, whereas the second uses a combination of ROS/Grasshopper/Rhino (RGR). The rudimentary feedback loops consist of a simple process where the robot is sent a motion instruction to execute an action. After this action is executed, the new position of the robot is communicated to the control source. With this new information, the control source determines the appropriate changes that are required to be made to the next motion instruction in order for it to be successful (Figure 12). This feedback process works in the same manner for both the isolated ROS environment and the RGR environment.

For the purposes of implementing this rudimentary feedback and for the future development of this research the, RGR feedback loop process has been established as the most viable process to achieve the outcomes of the research. The RGR process provides two primary benefits. This first is that the robot control functions are predefined, so this reduces initial complexity. Secondly, controlling the robot through grasshopper and rhino allows for a visual representation of the process the robot is undertaking to be established. This visual representation is particularly useful for troubleshooting issues encountered throughout the progression of the research.

As ROS only operates on Linux based operating systems, a method of communicating between the Linux based ROS system and the windows based Rhino/Grasshopper system is required. The methodology used for this is to develop structures which communicate through simple text files. The communication process consists of the Rhino/Grasshopper system writing a set of targets points to a text file which is then sent over a LAN to the ROS control structure for execution on the robot. The intel realsense camera, as described previously, is capable of generating a point cloud description of the environment. Using a text file based communication system allows for the coordinate data for each point in the point cloud to be written to a file and transferred from the ROS control structure to the Rhino/Grasshopper working environment, thus facilitating a feedback loop between the operating environment and robot motion execution.

This use of text files allows for direct communication between different operating platforms. Rhino is used in this process to better understand what outputs are developed from the intel realsense camera. A significant benefit of the text file communication system is the speed through which large amounts of data can be quickly transferred between systems. The initial system operated with a communication time of approximately 20 seconds however, with the implementation of a text file based communication system, this time was reduced to approximately 0.5 seconds. As a result of these improvements in communication speed, the text file communication system was implemented in all future applications of the feedback loop process.

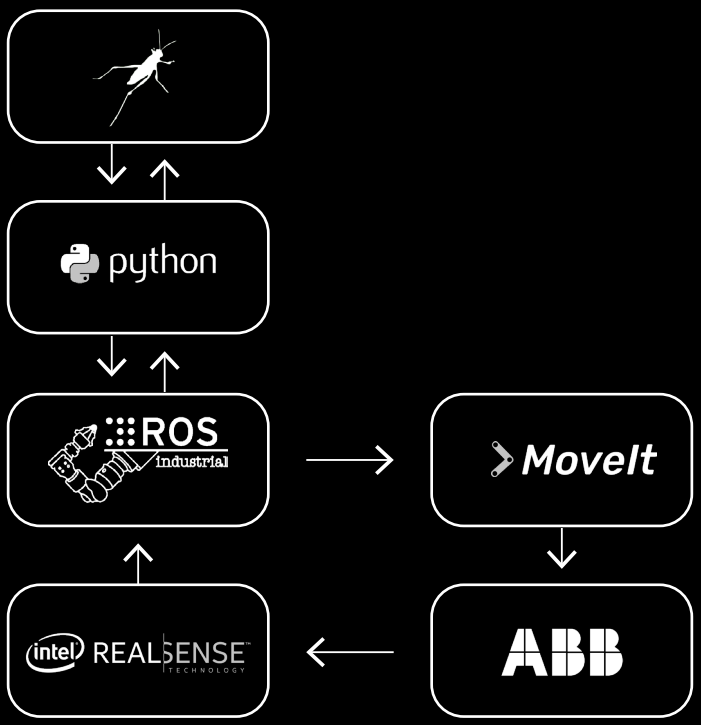

python
ROS
industrial
MoveIt
intel
REALSENSE
TECHNOLOGY
ABB

Figure 12 - Simple robotic feedback loop process

5.3 Environment information collection

Several different methods of environmental information collection were looked into during the initial investigation as to their viability in a feedback loop process including Computer Vision. Computer Vision uses a different output from the realsense camera, images, in order to determine the location of objects in the environment (Figure 13 & Figure 14). There are several limitations to this process which meant that it was discounted from being used in further prototypes. The difficulty in converting the image into terms that the robot can use to execute motion instructions was significant and more easily done from a point cloud. Current research also uses point cloud systems specifically for this reason, as they have far greater accuracy than that of an image, and can meet the low tolerances required in a construction environment process .

An additional limitation, when compared to the RGR approach, was the difficulty in accurately mapping the complexities of the robot operating environment. Computer Vision produces a pixel image of the environment under consideration by the visual sensor. This method of information generation contains significantly less information about the operating environment than is included in the point cloud description used by the RGR method. This lack of information reduces not only the accuracy of the block grasping process but also the ability for the system to discern between different elements in the environment, specifically the blocks required for construction of the wall system. This approach to object identification was discounted from future prototypes due to a lack of accuracy, difficulty in translating pixel information to useful data capable of translation to robotic terms and a lack of clarity in the information gathered from the visual sensor. The combination of these issues suggests that a RGR workflow is more appropriate to the scope of the research.

Another method that was considered during the initial investigations was Machine Learning. Machine Learning is a means of classifying outcomes from a large dataset for use in new and unknown situations. The Machine Learning process is most suited to applications that are after the feedback stage in a feedback loop process for example. Machine Learning

is suited to inferring relationships in the point cloud data set to identify the location of multiple objects with different geometries. The Machine Learning approach offers additional scope to the research that is best used after the successful implementation of a complex feedback loop system.

The inclusion of Machine Learning in the early stages of this research would have significantly reduced the research output. The complexity of implementing a Machine Learning algorithm is significant and not best suited as a means of achieving a simple feedback loop system. Machine Learning can however, provide significant benefits when used to begin making decisions for the robot. In this case, the machine learning algorithms can be manipulated into developing solutions that consider approaches not possible from a human perspective due to the size and complexity of the point cloud data set. Further investigations into the suitability of Machine Learning were conducted later in the design process.

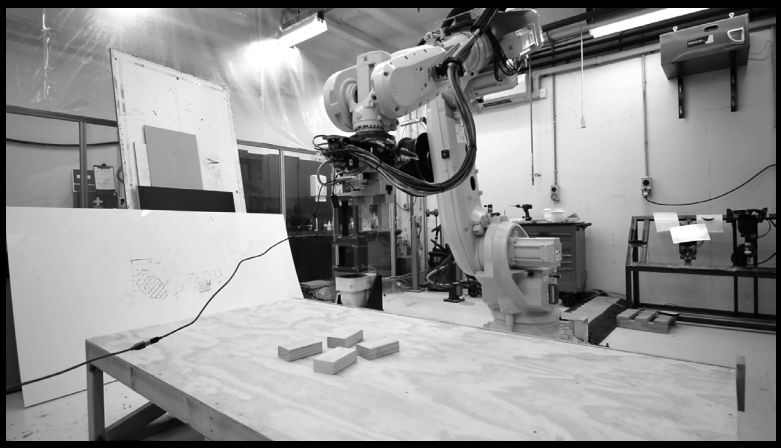


Figure 13 - The robot operating environment

Figure 14 - Construction Surface

6. Prototype One

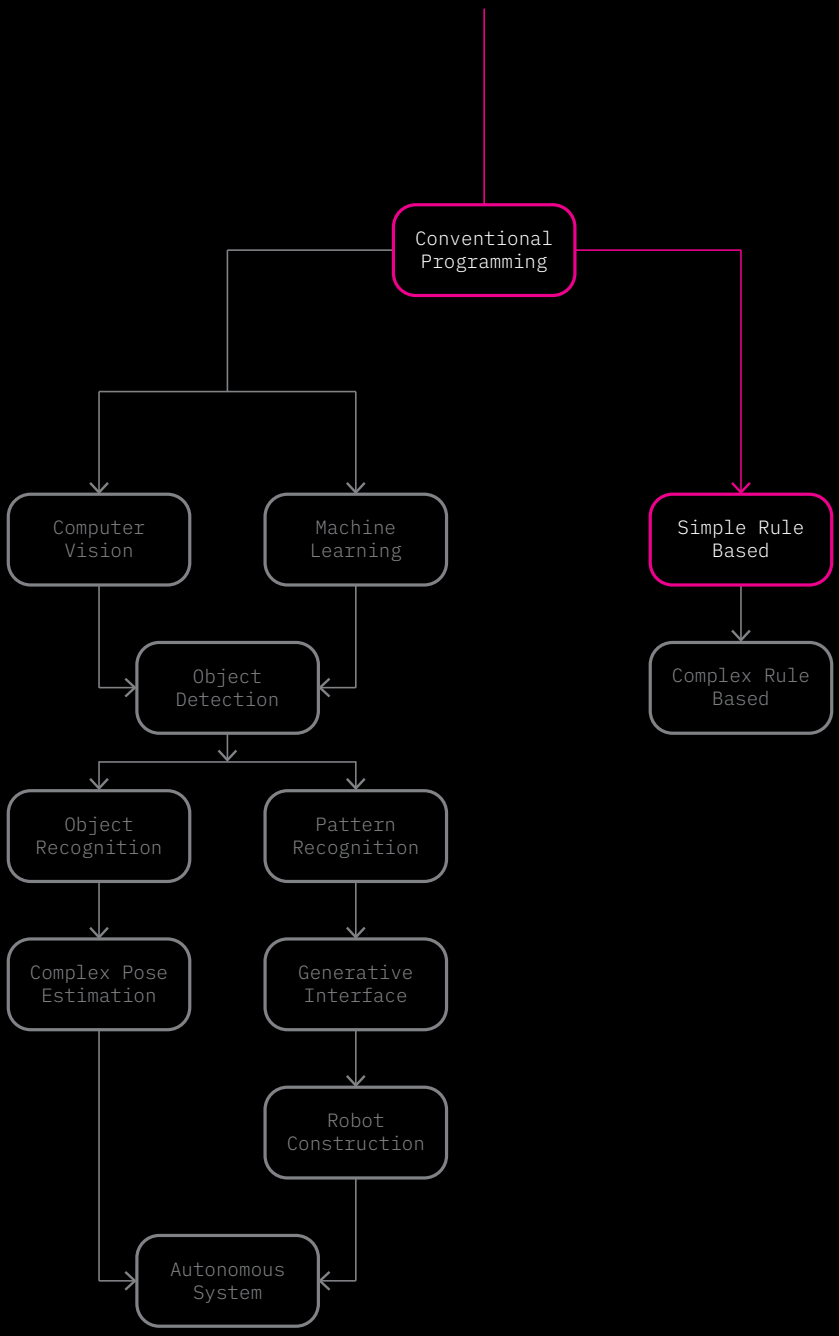

Conventional Programming
Computer Vision
Machine Learning
Simple Rule Based
Object Detection
Complex Rule Based
Object Recognition
Pattern Recognition
Complex Pose Estimation
Generative Interface
Robot Construction
Autonomous System

Figure 15 - Prototype One

The process of developing a feedback loop system relies on a robust methodology. All the steps in this process rely on a common architectural approach of iteration and prototyping where incremental changes are made to parameters and then analysed in order to determine whether such changes are useful or require a different approach to be undertaken. In contrast to most other research in the computer vision field, this research includes the use of Rhino as a means of facilitating interaction with a robot as a familiar interface for an architectural researcher. (Figure 16)

The first step in the process called for data transfer systems to be established in order to facilitate communication between the robot and Robot Operating System (ROS). ROS is an open source software package that facilitates live interaction with robots. ROS consists of two basic concepts that are useful to robotic feedback loops. Firstly, communication between nodes via what are called topics and secondly, the ability to easily extract data published to these topics. These two concepts allow for real-time communication to occur between a robot and an external CAD program such as Rhino.

Communication within ROS occurs through a series of nodes which either publish data to a topic or subscribe to a topic to receive data. The interface for this system is primarily through a high-level programming language such as Python. For the purposes of this research, Python was used due to the flexibility and adaptability it offers. Python also allows for development to proceed more rapidly due to a significant reduction in setup time compared to other programming languages. This interaction facilitates basic motion instruction communication between ROS and a robot with a ROS node that publishes point coordinate and motion instructions to a topic in which a subscriber node is collecting the instructions and executes them.

This concept can be further expanded and made more accessible through the use of Rhino as a visual interface to this process. Rhino along with Grasshopper provides a working method where architects can understand and define motion commands within an environment to be sent to the robot. This system is particularly useful for troubleshooting issues with robotic motion instructions as the visual feedback from Rhino allows

for an easier understanding of what instructions are being sent and the expected motion of the robot in a given environment. Rhino also facilitates defining the robot's environment in 3D. This presents an opportunity to reduce issues with collisions in the environment by using Rhino to prevent motion points being set outside of the robot's operating area.

As a result of ROS being able to communicate in real-time between a robot and a motion instruction, it is possible to use Rhino as a live interface between a human and a robot. Real-time interaction with robots is currently experiencing a significant uptake in research due to the possibilities of this type of robot human interaction. However, ROS itself is highly proprietary,code based and requires significant setup time making it difficult to be integrated into an already existing architectural workflow which is generally a visual one..

To facilitate the live connection between Rhino, ROS and a robot, a cloud-based server was used in order to seamlessly transfer files between different operating systems and devices. This can account for some slowdown in performance due to a file being sent over the network. Direct communication via ROS can be explored in future projects.

The next stage of the process required the setup and connection of an RGB-D camera. An RGB-D camera is required in order to generate both environment images and point clouds with associated depth information. An Intel Realsense D435 with ROS integration was used for this project due to its ability to produce a point cloud. This point cloud data was published directly to a ROS topic and then extracted to a plain text file through a ROS subscriber. This process was relatively straightforward with the data easily extracted and then exported into a readable format for integration into other parts of the workflow. By publishing to a plain text or comma separated value format, the data could be used easily in different iterations by different software packages.

The data extracted from the camera provided a three-dimensional representation of the environment in which the robot was located. From this data, differences between the original "clean" environment and

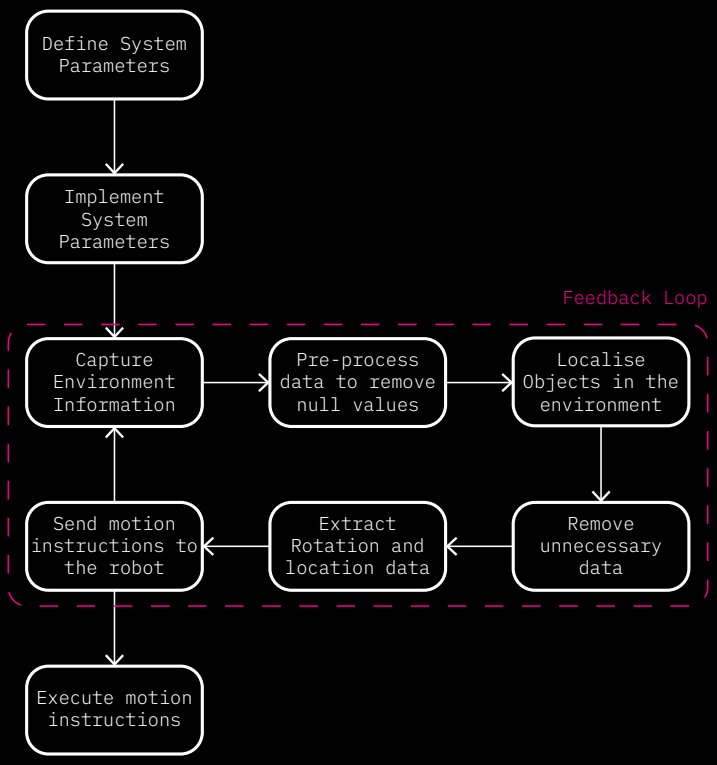
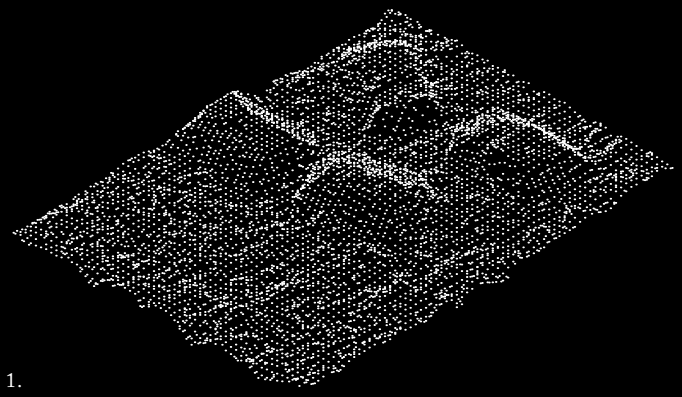
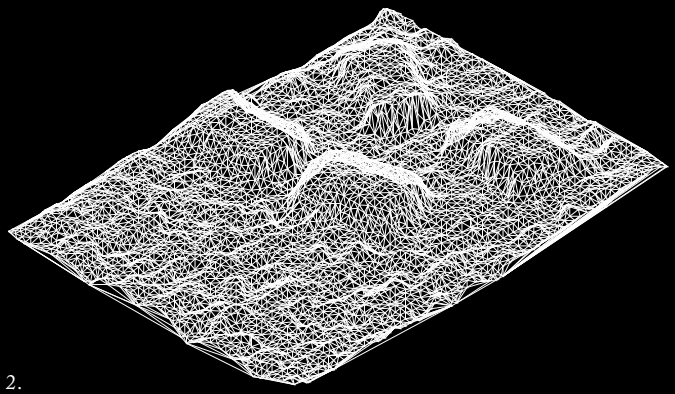

Define System Parameters
Implement System Parameters
Feedback Loop
Capture Environment Information
Pre-process data to remove null values
Localise Objects in the environment
Remove unnecessary data
Extract Rotation and location data
Send motion instructions to the robot
Execute motion instructions

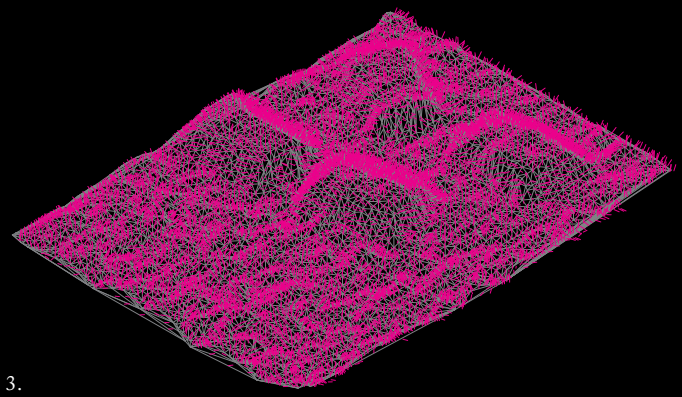
Figure 16 - Prototype One Functionality



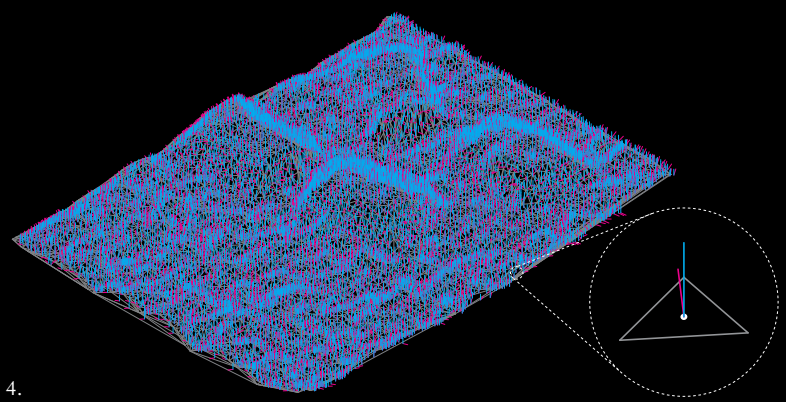
1.



2.



3.



4.

Figure 17 - Determining Mesh Normals

1. Base Pointcloud
2. Mesh from Pointcloud
3. Mesh face normals
4. Comparison of face normals to world vertical normals

— Mesh Face Normal

— World Vertical Normal

anything added to the environment could be compared to determine if anything new was in proximity to the robot. This was the first step to identify objects that could be moved and relocated by the robot. The process of transforming simple point cloud data into identifiable objects for the robot to move, requires significant data manipulation. Firstly, converting the data from a collection of points to a mesh is necessary to determine where the surfaces are in the data that is collected. Further to this, the normals of the mesh faces must be compared to a directly vertical normal in order to then determine which mesh face normals are perpendicular to the RGB-D camera (Figure 17). The next step is to filter these mesh faces based on distance from the camera in order to determine the top of an object's surface with reference to the robot frame.

The new 3D mesh now represents the highest surface within the camera frame, which is the top of the object. With this data, the mesh can now be matched to one of the predefined objects in the library. This match is based on a comparison between the area of the mesh and the area of the objects in the library. Mesh areas that were within a range of tolerance of the original object area were determined to be "blocks". The library object was then compared to the identified mesh in order to determine the location and orientation of the object and to provide a list of objects for the robot to manipulate.

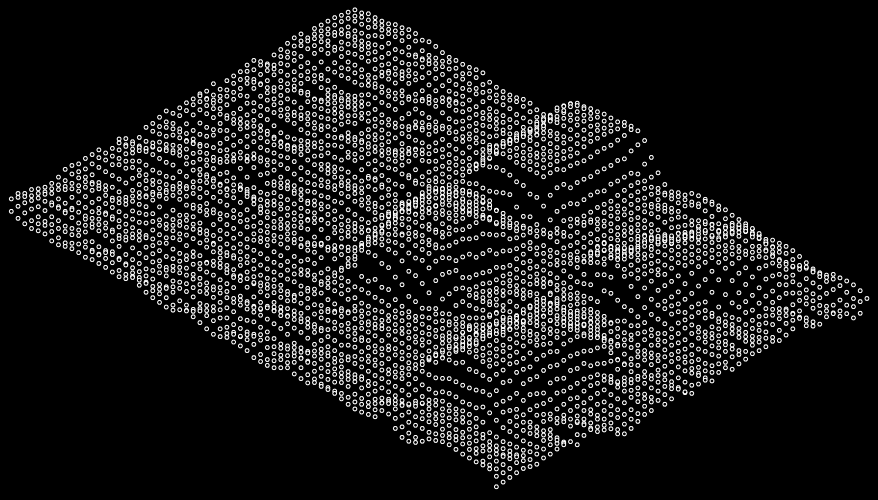
The amount of time required to identify a block relies on two key processes to be executed. The first process is the extraction of the camera data as a point cloud representation of the camera frame. The second is the organisation and manipulation of this data into a format that is understandable by the robot. The process initially required approximately 90 seconds to completely execute and output targets for execution on the robot. The processing time was a severe limitation on the effectiveness of the entire system. In order to improve the functionality, the amount of points in the extracted point cloud was reduced. This reduction occurred through subsampling the point cloud down from 30,000 to 3,000 points and had a profound effect on the overall processing time in order to identify a block, reducing this to approximately 5 secs.

Two robot motion operations were necessary for the success of the research. The first was to move and position the robot over the identified object and the second was to grab the object and place it in its final location. The identification and location of added objects to the environment of the robot was the first step to provide the necessary data for the robot to move and orient itself in reference to the object to be picked up. Through the identified mesh, the corner points of the block were determined. The corner points or extents are vital in establishing the position of the object in 3D space in relationship to the robot so it could be picked up and moved. The robot was then provided with instructions to move above and orient its gripper based on knowing the short and long side of the block.

The second motion operation, grasping of the object, requires the gripper to be located at the same position on every block in order to ensure the precise placement of the block. To determine the exact positioning of the gripper, the four corner points of the block need to provide two key pieces of information. The first is the centre point of the block, which is defined using two opposite corner points, and the second being the orientation of the block on a flat surface. This orientation is calculated using a vector which is parallel with the robot base y axis and a vector which describes the shortest side of the block (Figure 21). The angle between these two is calculated and transferred into a roll, pitch, yaw (RPY) and quaternion description of the block's orientation. Having determined these two pieces of information, the robot is now able to execute a motion instruction in order to grasp a block.

Finally, the robot was given instructions on where to place the block based on a predetermined location and the blocks were stacked. This part of the research related to previous work in pre programming the robot to move known objects to known positions.

The system developed through this research resulted in a process in which the robot was able to successfully identify the location and orientation of an object within a controlled environment. The system was able to determine the pose (Position and Orientation) of a simple rectangular block through the use of a library of object faces. These



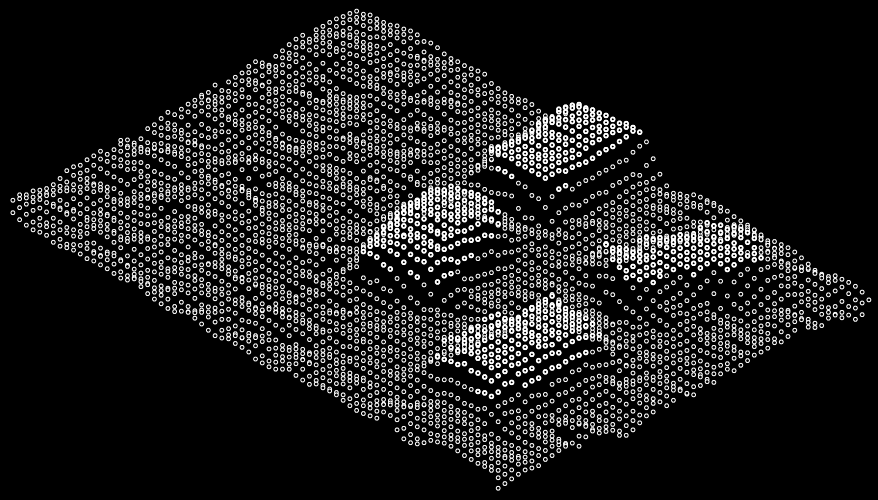
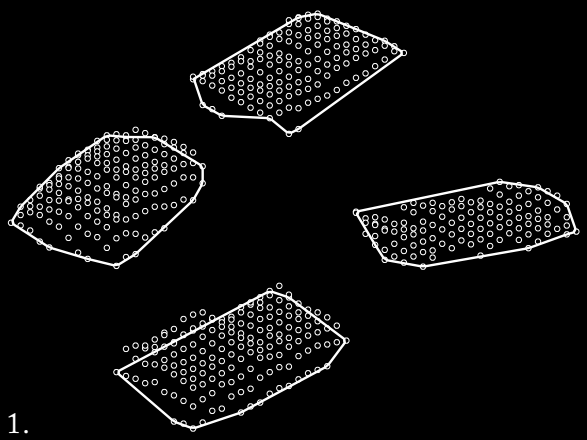
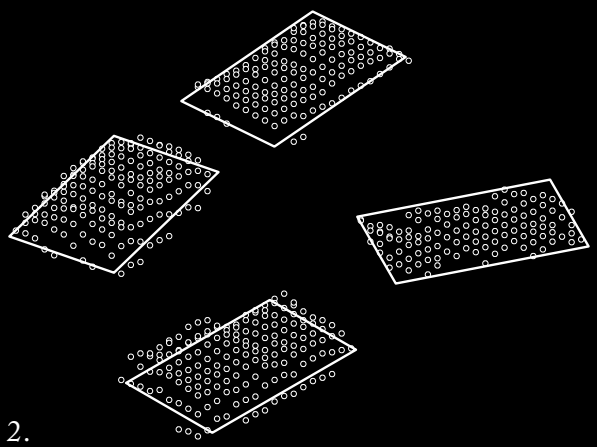


Figure 18 - Pointcloud Description

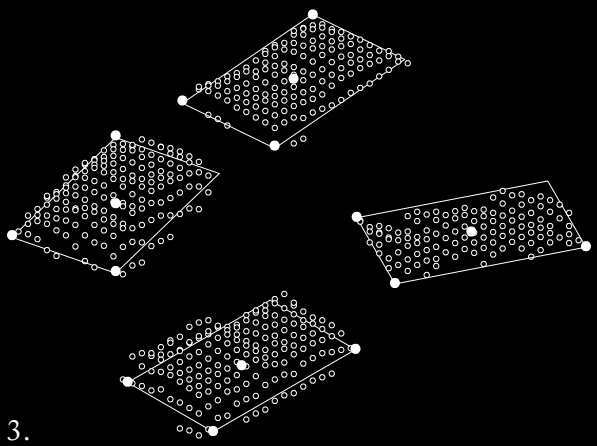
Figure 19 - Filter Block Positions



1.



2.



3.

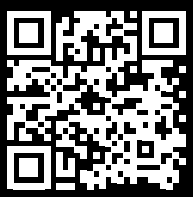


Figure 20 - Block Pose Estimation

1. Grouping Block surface points
2. Matching to Geometry Library
3. Extracting Corner and Centre Points

object faces were matched with the topmost surface of the block in the environment. The result of this process is a robotic feedback loop able to determine the extents of a randomly placed object from a known library of object faces that could be picked and placed precisely.

6.1 Limitations

The results of this research present a system for controlling robot motion through the integration of a vision sensor in a robotic feedback loop. When presented with a block/brick with known dimensions, the robotic feedback system was able to identify the location of the object in the environment, compute a set of planes to move to, and then execute motion instructions in order to move to the corner points of the identified object. The system was able to deal with the placement of the block on any of its planar sides, calculating the correct distance from the camera to the block and then executing motion instructions in order to detect the location of the given object in the environment.

Robotic feedback loops that employ vision as a means of object detection were successful with this research. The results have identified a series of restrictive limitations to be further explored. A fundamental limitation of this system is the speed at which the entire process is carried out. The process of manipulating a large dataset, along with the conversion of strings to floating point values, are key contributors to the reduced speed of operations. While this may not seem like an excessive amount of time, this execution time is only for a single object to be identified and very simple motion instructions to be executed. If the system was to be scaled to be more appropriate for the construction industry, this processing time could be significantly restrictive to widespread industry uptake.

Another limitation of this system is the accuracy of the extracted point cloud data. Robots operate with very tight tolerances, whereby a robot is significantly more precise than what the camera currently accounts for. When used in this system, the robot was lacking the critical accuracy that would be required to execute a pick and place motion instruction. The accuracy of the point cloud could be increased when manipulated in Rhino

and Grasshopper. By using a visual medium as a tool for understanding the point cloud data, minor adjustments and variations could be made to manage issues with accuracy. While this approach was moderately effective, further research would need to be conducted in order to improve the accuracy to match the tight tolerances expected in the operating environment.

Another less significant limitation is the restriction of object detection to a predefined library of object faces. In this research, the three different faces of the simple rectangular block where used as a library of objects in which a match between the resulting mesh geometry could be made. When considering the application of this research to extend further than it does currently, the limitations of this approach to object detection are more evident. When this system/process of object detection is transferred to an environment where the objects are not contained within the library, the lack of adaptability to new and differing environments containing objects that are unknown to the robot would cause this system to stop operating.

Robotic efficiency is best observed when a robot is undertaking a repetitive task and interacting with the same object over and over. The use of an object library in this research attempts to limit the complexity of the system in order to progress with further development beyond object identification. As such, the system is currently only able to operate and interact with objects that are rectilinear, and of which the geometry is simple e.g. blocks/bricks. This is a limitation, however, materials commonly used within the construction industry match these rectilinear limitations and therefore it is not something that requires an immediate solution in order to be effective as a means of unknown object detection.

The summation of these three limitations is that while the research is relevant, when considering approaches for further investigation, different means of locating and detecting objects in the environment will require an effective means of manipulating visual data and an increase in computational efficiency to provide a more reliable means of operation in the long term. While this system is limited, there is the potential for a number of different approaches to be taken to further

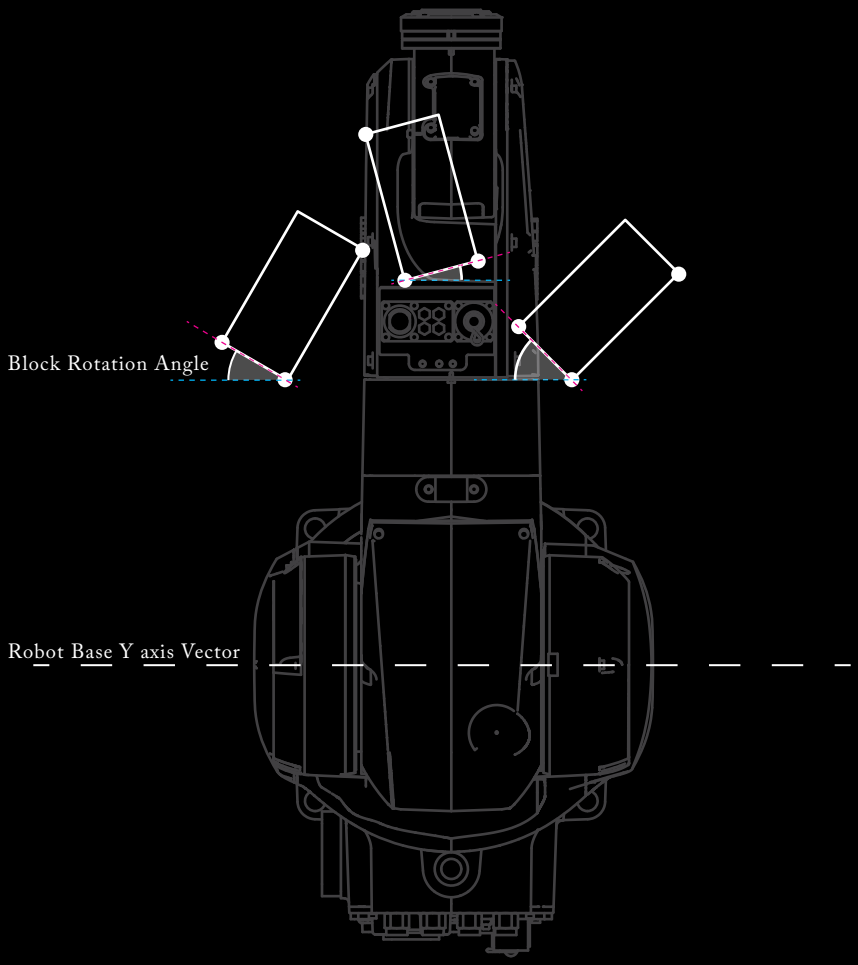

Block Rotation Angle
Robot Base Y axis Vector

Figure 21 - Block rotation angle calculation

--------- Block Short side Vector

--------- Robot Y Axis Vector

the research (as outlined below). These different approaches have the potential to develop this system of object detection to the point where it can be integrated with completely unknown environments and objects.

This research has been successful in identifying objects within a controlled environment. To progress further there are several opportunities to pursue that could provide solutions to the limitations of the system outlined above. The first such opportunity is to investigate the potential for an image-based object detection system to operate to the same or better extent than the current system. Computers are more efficient in processing bitmap image data than they are in dealing with the conversions of text to numbers. As such, a system that utilizes this processing power could operate at a greater speed than the current system with the potential for a real-time system to be developed. Another potential avenue to examine is the use of improved hardware for kinematic planning and collision detection. Sorin and Konidaris (2018) have developed a system whereby kinematic planning and collision detection is undertaken in milliseconds, rather than the 1 - 2 seconds currently seen with basic consumer-grade software. Finally, the integration of camera calibration into the system provides an opportunity to improve the accuracy of the extracted point clouds which in turn will increase the accuracy of robot operations within the operating environment.

7. Prototype Two

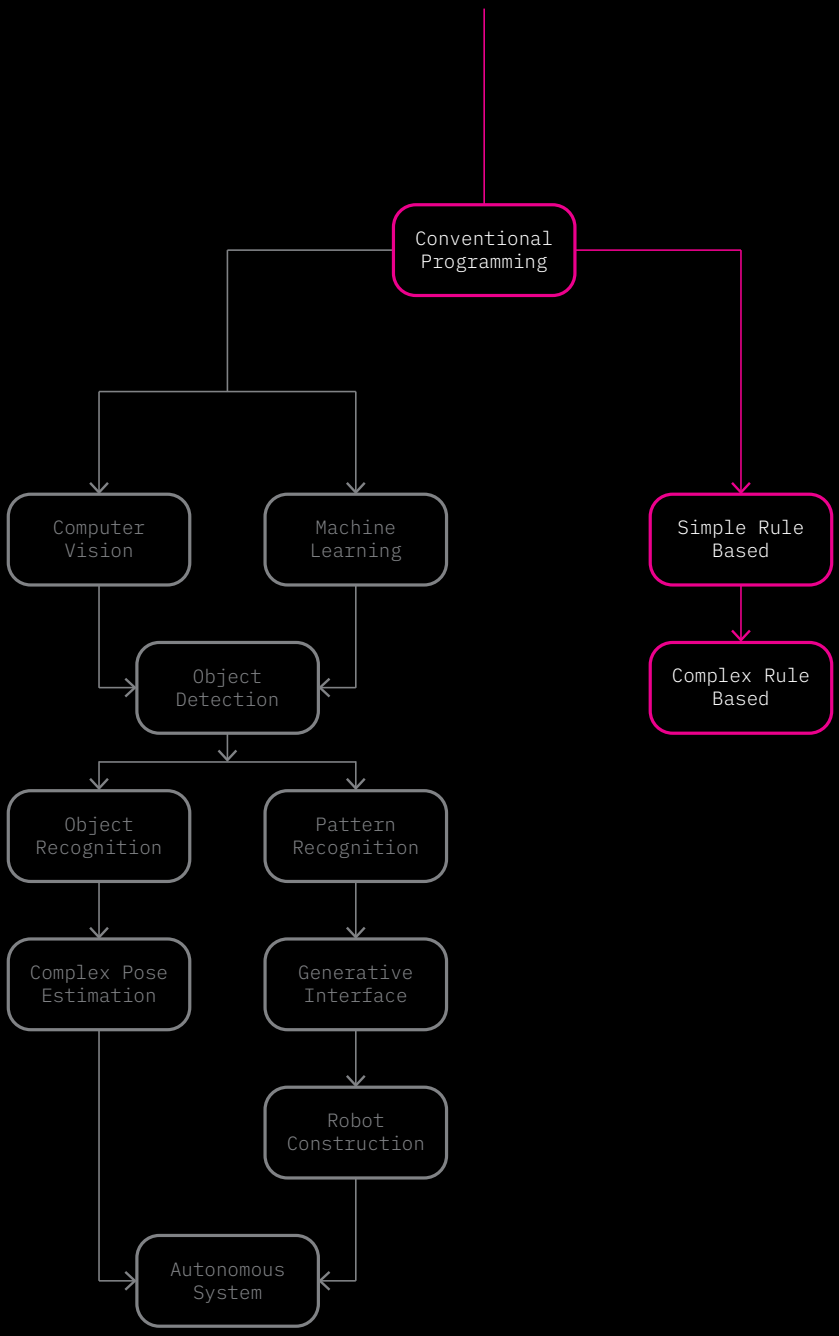

Conventional Programming
Computer Vision
Machine Learning
Simple Rule Based
Object Detection
Complex Rule Based
Object Recognition
Pattern Recognition
Complex Pose Estimation
Generative Interface
Robot Construction
Autonomous System

Figure 22 - Prototype Two

The second prototype builds on the previous prototype by integrating additional complexity to the robotic pick and place operation. The system functions in a broadly similar way to the previous prototype but with new systems included to control sequential object grasps, different wall construction locations and improve grasping accuracy. These three issues have been identified as the key restraints in the previous prototype, specifically the inability to build a wall with more bricks than could fit into the visual sensor frame. The functionality of this system follows a capturing of the operating environment in digital terms, appropriating this data to align with robot instruction sequences, and redirection back to the robot for execution. (Figure 23)

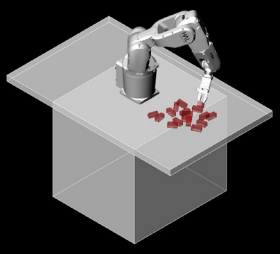
The goal of this prototype is that the robot constructs a simple block wall using blocks found in the operating environment. The position and location of these blocks are unknown to the robot until information from the visual sensor is processed to determine where in the operating environment they are located. Functionality to repeat the information input process will allow for the construction of larger and more complex wall structures. Improving the accuracy of the block grasping is also an essential goal of this prototype.

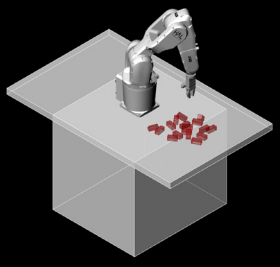
The second prototype functions by first initialising several different systems that deal with background processes outside the primary pick and place operation. The first of these provides the robotic system with a means of retaining information concerning the state of the currently constructed wall. When first starting the system, the user-defined wall is translated into a list of block placement coordinates. These coordinates provide the robotic system with a description of all the blocks in the wall. This information is stored in a CSV file. Throughout the pick and place operation, the system updates the CSV file when the robot successfully places a block, thus providing an up to date description of the progress of wall construction in digital terms. The second initialisation process involves communication between the robot control systems and the robot. As the previous prototype demonstrated, after communication between the two exceeded 70 unique communications, the system failed to function correctly as the grippers failed to either open or close. In order

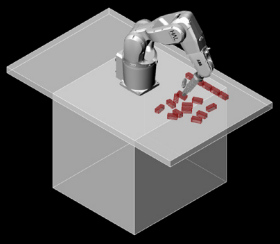
to overcome this limitation, a 'cookie' received from the robot during the initial HTTP request is re-sent with each subsequent request to maintain the link between the robot control systems and the robot. Both of these systems address issues with the previous prototype, thereby increasing the ability of the robotic system to complete more complex tasks.

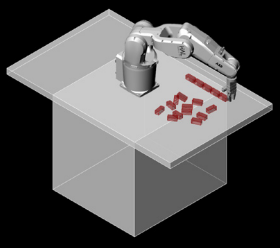
Environment information is captured through the robots visual sensor before the wall construction process begins. Critically, as discovered in the previous prototype, the distance between the ground surface and the visual sensor is essential in gathering accurate information about the state of the operating environment, specifically the position and orientation of the block objects. If the visual sensor is too close then not enough information is available in each image capture and if too far away, then the accuracy of the resulting point can be off by as much as 20mm. A distance of 400mm from the ground surface to the visual sensor lens appears to be most effective at balancing the two constraints. The output from the visual sensor describes the operating environment as a Point cloud which is transferred from the visual sensor to the robot control system for processing.

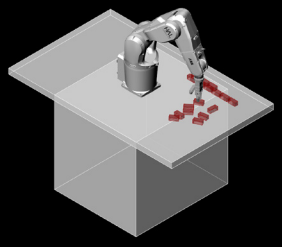
The Point cloud export from the visual sensor contains a point representation for each pixel in the image and as such, for an image with a resolution of 1,280 x 720 pixels, the Pointcloud contains 921,600 individual points. In an attempt to improve the processing times this initial point cloud was subsampled down to 10% of the original size while still maintaining the structure of the pointcloud. This subsampling improved the overall processing performance significantly. In addition, the speed of communication between the robot and the robot control system has also been improved by transitioning from a cloud-based communication service to a local network protocol. While the cloud-based service was taking approximately 20 seconds to complete communication processes, the local network solution has reduced this to approximately 0.5 seconds. Switching to a local network file transfer has also allowed for significant improvements to the complexity of the initial point cloud that can now be transferred from the visual sensor to the robot control systems.

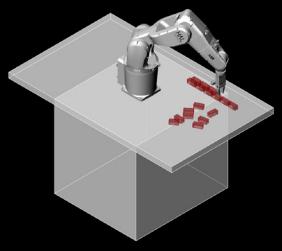
1. 

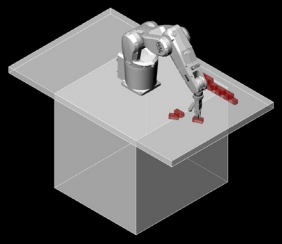
2. 

3. 

4. 

5. 

6. 

7. 

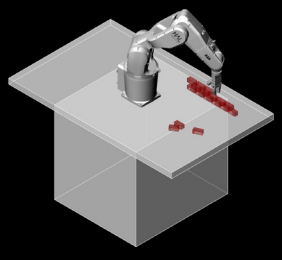
8. 

Figure 23 - Autonomous robotic grasping sequence. The robot grasps the randomly distributed blocks to construct a simple block wall.

In contrast to the initial prototype, this system uses a new method for determining the position and orientation of the blocks in the visual sensor frame. The initial point cloud is filtered by height above the ground surface, in a similar way to the initial prototype however, this is where the similarities conclude. In the initial prototype, the rotation angle of the block often failed to be correctly calculated whereas this prototype integrated an inclusion test. The system uses a corner point of the block, typically the corner point with the lowest x value coordinate,to test for the inclusion of two other corner points within a specific radius.

After the conclusion of determining the position and orientation of the blocks, this information needs to be translated into data that the robot understands. From each of these poses, the centre point of the block along with the angle of the block in relation to the robot act as inputs for the motion instruction generation system. The rotation angle of the block controls the rotation of axis 6 of the robot, the grippers, aligning the grippers with the block to execute the pick and place operation.

After the creation of the block position and orientation instructions, the control system transfers these to the robot through the ROS communication system for execution. The robot will fully execute an instruction sequence for the number of blocks present in the visual sensor frame at the initial point of capture. If the blocks were to move during this instruction execution process, the robot would be unaware of this and continue as though the blocks were in the original positions. After this instruction sequence, the robot returns to the initial capture position to repeat the process and add to the already existing wall rather than begin constructing a new wall.

7.1 Limitations

This prototype makes a significant number of improvements to the first prototype. There are still several limitations present in this prototype that are restrictive in allowing the system to undertake more complex pick and place operations. However, this prototype still has significant potential to be developed in its current direction in future research. The combination of developing systems to remove the limitations in the system

and implementing potential system improvements offers the opportunity to develop a complex robotic system capable of dealing effectively with varying operating environment conditions. These limitations and potential opportunities will be discussed in more detail with a particular focus on why the described limitations are inherent in a system of this type.

The limitations of this prototype can be organised into three main categories; Operating Environment, Lighting and Technical. Solutions to the first two limitations are more easily found by making physical adjustments to the operating environment and the systems operating within it. The technical issues limiting the progress of this research however, are more complex and require a significant increase in both time and resources to overcome. As such, attempts to improve the operating conditions have been more thoroughly interrogated and explored compared to the more arduous technical aspects of the system.

The state and setup of the operating environment were fundamental to the success of this prototype. Three critical conditions were essential in ensuring the successful operation of the system. Firstly, the positioning and texture of the ground surface of the operating environment is critical in ensuring high-quality data input to the visual sensor. During the prototyping phase, it was found that highly reflective surfaces reduced the effectiveness of the visual sensor to map the operating environment with a high level of accuracy. The lack of accuracy is a significant limitation of the system that then requires the operator to have a reasonable amount of control over the environment in which the robot is operating. In applications that cannot be controlled, such as on a construction site, the ability to maintain a consistent surface will be limited. While this limitation was able to be successfully rectified in a laboratory environment further applications of this research will need to consider the implications of surface materials on the quality and accuracy of visual sensor data.

Control of the lighting conditions within the operating environment was equally essential to the success of the pick and place operation. Poor lighting conditions resulted in the blocks within the operating

environment casting sharp shadows. These shadows were commonly interpreted by the visual sensor as block edges resulting in blocks being ignored by the system or the position and orientation of the blocks being incorrect. Several methods were used in an attempt to overcome these issues. Multiple lights were set up around the visual sensor capture area to remove shadowing from a single direct light source. This approach was highly effective in reducing the number of 'misreads' by the visual sensor significantly. However, this approach is not transferable to different operating environments. In an attempt to address this, a light was attached to the gripping tool on the end of the robot. This approach was slightly less practical than the consistent illumination, however it offers significantly more flexibility to the robotic operating system. Overall, lighting is a significant limitation of this prototype and has been addressed to improve the accuracy of the overall pick and place operation.

Several limitations exist that relate to technical aspects of the robotic system. Firstly, the accuracy of the wall being constructed is reduced primarily due to the grasping of the blocks, which is often inaccurate in the long direction of the block resulting in uncontrolled wall construction. Several different approaches were taken in an attempt to improve the accuracy of the robotic system. Firstly, the algorithm that filters the point cloud into each block was interrogated in order to determine where the issues with the accuracy of the system may be occurring. Review of this process meant that an entirely new approach to the sorting of data gathered from the visual sensor was required. Secondly, as described previously, significant attempts have been made to improve the accuracy of the visual sensor data through controlling the physical operating environment variables.

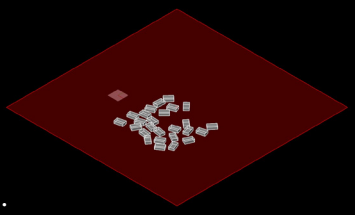
The second technical limitation is that the robotic system can only pick-up blocks that are parallel with the ground surface; blocks on angles off of the surface are unable to be identified. This requires a fundamental shift in the technologies used to determine the position and orientation of blocks in the operating environment by transitioning from a typical programming workflow to a machine learning-focused solution. Given the complexity of machine learning, transitioning to this approach is outside the scope

of this research however portions of machine learning approaches have been integrated to slightly improve the abilities of the robotic system.

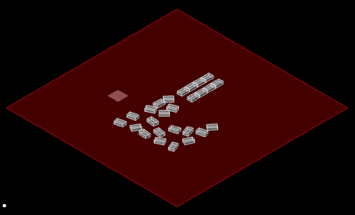
Thirdly, the robotic system sits in a loop executing a scan every 8 seconds while no blocks are detected in the camera frame. The length of time between loops is restrictive as the camera requires a certain amount of time to start and stop capturing. A potential solution to this issue is a system that is continuously capturing data through the visual sensor. This will reduce the slowdown created by multiple camera scans and improve the robot's knowledge of the operating environment. The technical limitations of this prototype are significant and if this research was to continue there is potential for these issues to be adequately addressed to improve the capabilities of the robotic pick and place system.

The prototype has several areas that are suitable for further research and investigation. Firstly, there is the potential to cover a larger scan area by having multiple visual sensor capture locations. Moving the robot to unique predefined capture locations would allow a more significant portion of the operating environment to be used to pick blocks from. Secondly, there is the potential to build multiple walls simultaneously and remember the state of each wall thus allowing the robot to multitask using the resources available to it. Thirdly, there is the potential to decide which block should be used with each wall based on distance from the block to the wall. An algorithm could be used to stipulate that the distance between a block and a place target should be as small as possible. The combination of these improvements would provide the robotic system with significantly more ability to construct complex structures.

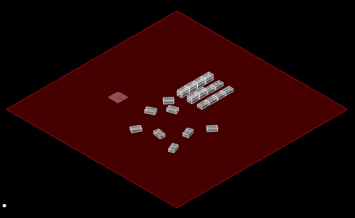
1.



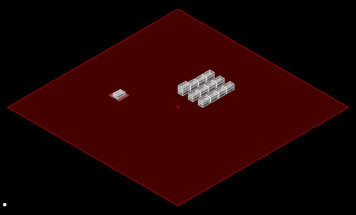
2.



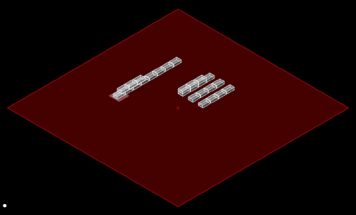
3.



4.



5.



6.

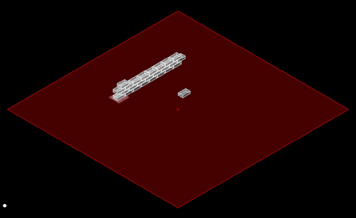


Figure 24 - Pre sorting randomly distributed blocks to improve wall construction accuracy.

8. Prototype Three

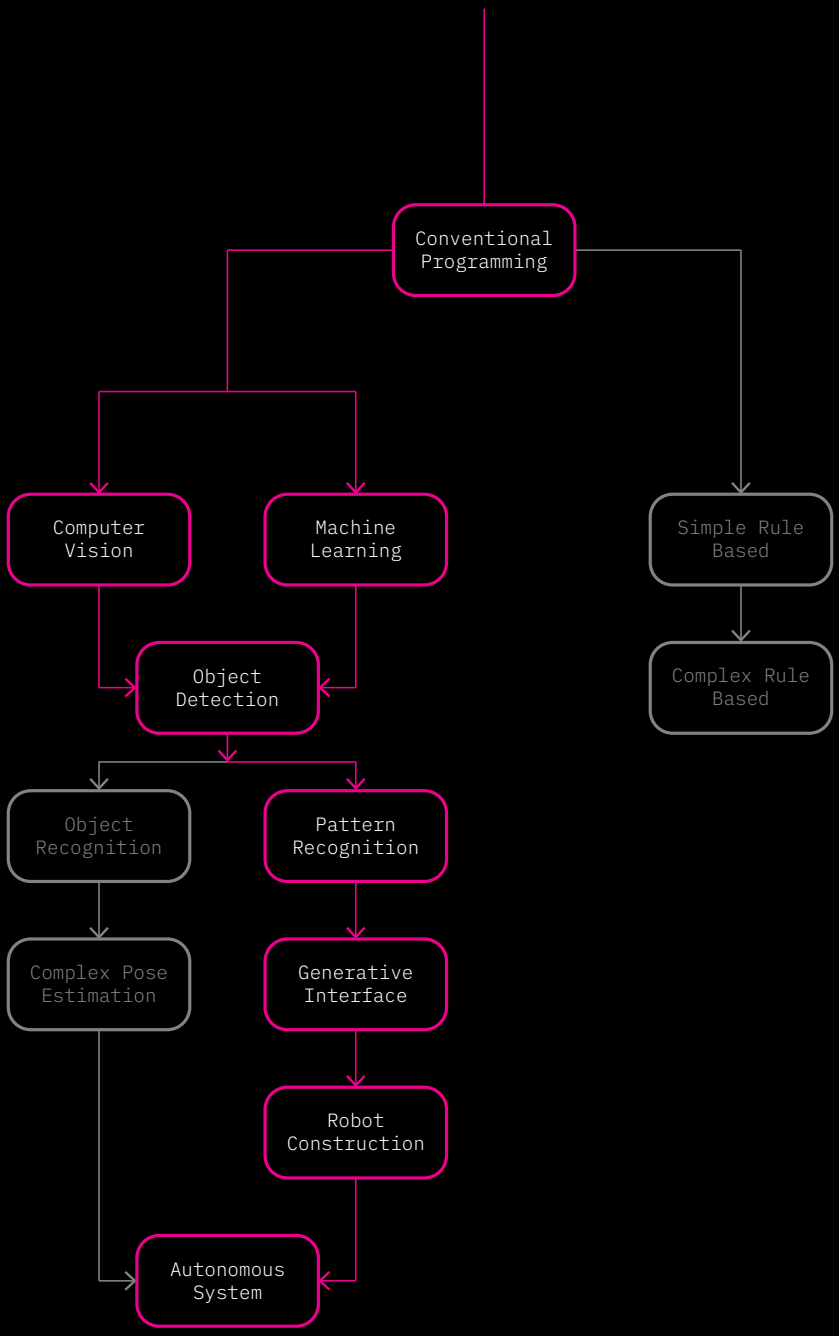

Conventional Programming
Computer Vision
Machine Learning
Simple Rule Based
Object Detection
Complex Rule Based
Object Recognition
Pattern Recognition
Complex Pose Estimation
Generative Interface
Robot Construction
Autonomous System

Figure 25 - Prototype Three

Based on the results of the two previous prototypes, the limitations of each are complex and will likely require substantial technical investment in order to provide a viable architectural solution. The difficulties outlined in the previous prototype, specifically the difficulty in accurately grasping blocks which locations are not known, suggests reverting to the well established process of grasping blocks from known locations. While simplifying the grasping process may seem to be a backward step for the research this prototype implements a machine learning algorithm that is capable of determining the extent and content of an image presented to the robot through the imaging sensor. The machine learning process offers significant complexity to the prototype in conjunction with developing the autonomy of the robotic system from a different perspective with the intent for future research to combine the two approaches.

The new system works in a process similar to the previous prototype. A stack of blocks is arranged by the operator, the location of which is either pre-programmed or input by the operator. The block stack arrangement removes the issue of accurately locating the block through the use of the visual sensor. The lack of accuracy in grasping the blocks in the previous prototype is thereby reduced by using the operator to control the robotic operating environment, thus simplifying the complexity of the conditions in which the robot has to operate. Using the operator to pre-locate the position of the blocks for the robot to grasp allows the overall system to be used to undertake more complex operations without being restricted by a lack of accuracy in the block grasping and placement process.

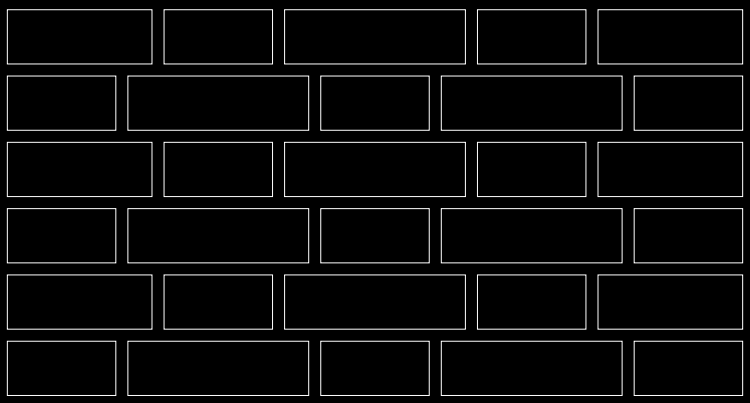
The operator either self generates or allows the program to generate a block wall pattern (Figure 26). The pattern generator program will generate a random arrangement of two different block orientations, primarily for simplicity for the initial testing stages. The operator inputs the length and height of the wall that they want the system to construct. The system, through the use of a machine learning algorithm, determines the required number of blocks for each row and the total number of blocks required for the overall wall. The pattern generation is formulated through a random choice algorithm where the length of each row is divided by the

number of blocks in both orientations to generate a random wall pattern. Once the wall pattern is created the program generates a pixel line image of the block wall which is exported for use in the robotic operation.

Once the wall pattern image was generated, the process of physically constructing the wall in the robotic working environment can begin. The first step in this process consisted of transferring the information generated in the block wall pattern image into digital terms that the robot understands (Figure 27). The process begins by having the operator present the image to the robots visual input sensor (intel realsense D435). The robot captures an image of the block wall pattern to use in a sequence of algorithms that determine the parameters of the block wall to be constructed which are then translated into motion instructions for the robot. The first of these algorithms is a Canny edge detection algorithm. This algorithm filters distinct edges from the background of an image allowing the edges of each block in the wall to be determined and translated into digital terms. This first step translates physical plans into digital information that can be manipulated for future use.

Once the edges of each block have been determined, the system introduces a second algorithm to determine the face area of each block which is calculated using the pixel coordinates of each continuous block edge. These areas are matched to the actual value of the block face area, either the end or side of the block, in order to determine the wall pattern. This information is stored in an order list, with the first item being the first brick to place and the last item being the final brick to place.

The motion instructions for the robot's actions are generated from this list of blocks. As this system uses only two different types of block orientations, the motion instructions for each block placement in the wall construction are to either place the block long-side parallel to the wall direction or, the end of the block parallel to the wall direction. Only two different motion commands are then required in order to place the blocks in the correct position within the wall structure. The motion instructions consist of the same instructions that are present in



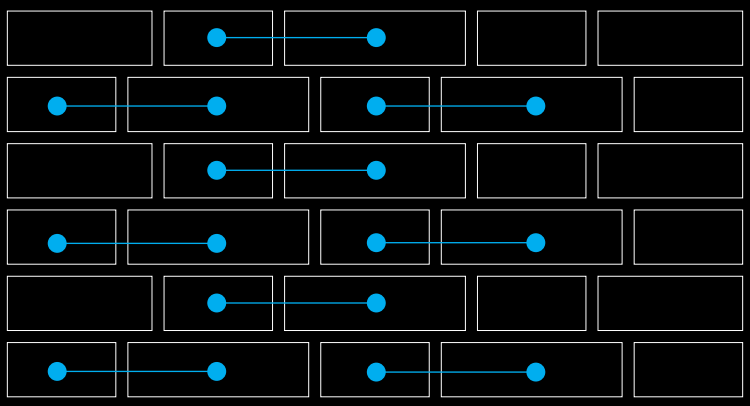


Figure 26 - Pixel image of wall pattern used to instruct the robot

Figure 27 - Machine Learning through pattern matching and analysis

the previous prototypes where the communication between the robot control systems and the motion instruction algorithm system is facilitated through ROS. This process of communication ensures that additional functionality can be added to the system in the future without the communication process hindering the capabilities of the robotic system.

After the motion instructions for the placement of the blocks in the wall structure have been generated, the predefined motion instructions for the grasping of each block in the pile of blocks to use is weaved into the list of motion instructions in order to have the robot correctly grab blocks from the pile and place them in the wall structure. The operator is in control of the process of preventing the robot from colliding with the operating environment. This presents a number of limitations in this prototype that restrict the comprehensiveness of the system to manifest complex architecture construction situations.

Overall this prototype functions through predefining the location of a pile of blocks to use in the wall construction and generating a random or user-defined block wall pattern for two different block types. Physically presenting the robot with a drawing of the block wall pattern and, using a combination of three algorithms allows the robot to determine what the pattern of the wall is. From the pattern determination, motion instructions are generated which allow the robot to execute tasks that involve grasping the blocks in a pile and then correctly placing the blocks to create the wall described in the two-dimensional pixel image.

8.1 Limitations

This prototype has several limitations, primarily revolving around the complexity of implementing digital accuracy in the robotic operating environment. Firstly, the blocks are picked from known locations. Secondly, the system is constrained by the type of image that can be used to instruct the robot. Thirdly, as in the other prototypes, the lighting around the robots visual sensors is imperative in achieving accurate identification of what the robot is seeing. Finally, the robot is unaware of the extent of the operating environment. The combination of these limitations will be discussed further

with an emphasis on potential solutions if the research was to continue.

The first limitation of this prototype is the requirement for the position of the blocks to be explicitly programmed into the system. This has two restrictions; firstly the blocks must all be positioned in a uniform manner in the same plane as the operating environment ground surface. Secondly, the accuracy of locating the position of the blocks in relation to the robot is critical in ensuring the accuracy of the resulting wall structure. This requirement shifts a significant amount of responsibility for the accuracy of the system to the operator. While having the operator position the pile of blocks may work in a controlled laboratory environment, the introduction of other variables typical in construction infer that the robot needs to be capable of finding the blocks in the operating environment itself.

The second limitation of this prototype is the limited scope of images that can be read into the algorithmic system. As the system is designed to read simple pixel line images, presenting the system with an image that does not fit the specific requirements of the system can cause unforeseen errors. The system will still attempt to process the visual information however, the outcome will cause the robot to operate in an unexpected way. The reason simple pixel line images were used to instruct the robot is the ease of distinguishing between black and white pixels in the edge detection process. In order to increase the complexity of the image that can be read by the robot, without reducing the accuracy of the resulting output, requires the integration of a filtering algorithm to remove unnecessary information from the image before running the edge detection algorithm. This creates a significant limitation however, it does not fundamentally restrict the functionality of the system in successfully constructing the block wall structure.

The third limitation of this prototype is the requirement for operating environment conditions to be perfect when the visual sensor is in operation. As discussed in the previous prototype, the lighting of the area that the visual sensor is viewing is required to illuminate the surface uniformly. However, given how the image of the block wall pattern is

presented to the robot, a small amount of shadowing is always present on the image. This can cause some confusion to the edge detection algorithm but can be resolved by merely including a filtering algorithm in the image processing system to reduce the harshness of the shadow in the images. The limitation was promptly resolved and presented little restriction to the functionality of the overall prototype system.

The fourth limitation of this prototype is the lack of operating environment awareness from the robot. The robot is unaware of the state and the objects that are present in the operating environment. This is particularly limiting when errors occur in the robot environment interaction. As the robot cannot perceive what is happening it has no way to anticipate potential errors occurring. As such, preventing collisions with the wall being constructed or the pile of construction materials are the responsibility of the operator to both foresee and act quickly enough to prevent. The position of the visual sensor at the end of the robot limits the ability to use this device to act as the eyes of the robot and provide updated information about the state of the environment, due to the potential for this visual sensor to be obstructed by objects held in the tool attached to the robot. A viable solution to this issue would be to employ an additional visual sensor however, both the cost of an additional sensor and the requirement to calibrate two sensors so they act correctly, are significant undertakings but still present the best method for integrating operating environment awareness in this prototype system.

To overcome the limitations discovered through the development of this prototype requires a combination of the system developed in prototype two to be integrated with prototype three in order for the robot to establish where the initial positions of the blocks are and the number of blocks that can be used to build a wall. Additionally, the development of prototype two to a stage where blocks placed in any position or orientation can be grasped accurately offer the potential for this prototype to act in an almost autonomous manner. The inclusion of elements of the work currently in development by Gramazio and Kohler, discussed in a subsequent paragraph offer the potential for the development of a robotic system capable of operating autonomously on site.

The potential of this prototype is to allow a robot to read a set of plan documents or similar instructions for a construction project. Having read these plans, the robot can determine where on-site it needs to be to complete a construction task, what tools it needs to complete the task, what materials it needs and also the sequencing it needs to place the materials in order to complete the construction task. The robot can act as though it is a competent human construction worker, interpreting drawings and then constructing the building on-site with the added advantages of repeatable accuracy, 24/7 operation and significantly increased operating capacity.

Looking within the literature at the current state of the art suggests the potential, as alluded to earlier, to integrate aspects of both this research and the work of Gramazio and Kohler. Gramazio and Kohler's work focuses on providing some awareness of the operating environment to the robot, a fundamental limitation of this research due to both time and cost constraints. Combining the two, along with the interpretation of construction documentation by the robot and an accurate understanding of the environment the robot is operating in, begins to bring about the overall potential of this system to act as an alternative for the on-site construction worker.

9. Discussion

9.1 Linking the Research to the Literature

The practical outcomes of this thesis have demonstrated the beginnings of a complex robotic system capable of executing basic construction tasks without human intervention. It is pertinent with this new knowledge to reinterrogate the literature with a specific emphasis on situating this thesis within the existing body of knowledge. Linking the outcomes of this thesis to already existing work, formalises the conclusions drawn from this thesis. More importantly, it aids to reinforce the relevance of the system to both architectural theory and practice. The specific areas of interest will be discussed further through two lenses. First, a theoretical examination followed by a practical analysis of the conclusions of this thesis.

There are several areas of interest when comparing the outcomes of this thesis to the literature. Firstly, a comparison with intelligence theory and an evaluation of the conditions that define intelligence against the outcomes of this thesis. Secondly, a critical analysis of the implications of perception within the scope of this thesis. Thirdly, autonomy is examined with the intent of determining the level of autonomy present in this system. Fourthly, the process of decision making is examined through the lens of self-decision making by the autonomous system. Finally, the process of learning will be examined with a particular weight given to fundamental learning techniques within artificial robotic systems. The combinations of these theoretical positions serve to cover the relevant literature within the scope of this thesis.

9.1.1 Intelligence

This thesis presents intelligence as the ability to interact with environments and carry out adaptive, goal-orientated objectives. The questions relating to the definition of intelligence discussed in chapter two are a useful means of presenting the outcomes of this thesis in terms of being an intelligent system. Examining the first question; Is intelligence the level of competency displayed when undertaking both unknown and familiar tasks? With regard to the outcomes of this thesis, it is clear that the system proposed in this thesis displays some level of intelligence. To be able to interact with the environment, even in a rudimentary manner, suggests that the system

displays a basic form of intelligence. The second question, Is intelligence the rate at which a system can learn? This thesis does not explicitly encounter a rate of learning indicator, as the system is primarily focused on achieving a singular task and as such learning was not a primary driver of the research direction. However, functionality is included in the system which begins to exhibit learning like behaviour where the more attempts the system makes at a particular task the more competent the system becomes at that particular task. Based on the formerly presented definition of intelligence it is clear that the resulting robotic pick and place system developed through this thesis is beginning to exhibit a rudimentary level of intelligence across the spectrum of intelligence metrics present in the literature.

9.1.2 Perception

Perception in the scope of this thesis consists of two layers of complexity which aid in defining a manifestation of intelligence in artificial systems. Firstly, environment perception simply translates to reproducing the physical environment in digital terms. The second layer is significantly more complex allowing an artificial system to prepare itself to interact with any environment. Primarily this is a preconceived response developed through the system's experience that assesses the usefulness of objects within an environment in assisting the system in achieving the required task. The resulting system of this thesis operates primarily within the first layer of perception, merely gathering a mapping of the environment to be processed at a later date.

The difficulty with implementing the second layer of perception is significant. In order to preconceive a response to specific situations and objects within an operating environment, the artificial system must have significant experience operating in a multitude of different environments. The aim of addressing complex perception was outside the scope of this thesis due primarily to the complexity of developing such a system. There were also a lack of existing attempts as a reference point for the development of a perception system capable of pre-emptive action to meet a target or goal. The resulting system of this

thesis attempts to begin to integrate the idea of perception with having the system undertake a series of predefined instructions that assist in allowing the system to preemptively understand the environment.

9.1.3 Autonomy

Autonomy, defined in this thesis based on a comprehensive analysis of the literature, refers to the ability of an artificial system to achieve a particular task. When considering autonomy in this manner it is clear that the resulting system of this thesis has achieved a level of autonomy. This level of autonomy is restrictive as the robotic system still requires a significant amount of human intervention to complete simple tasks. This human intervention takes the form of control over the environment and preventing the robot from making errors critical to the successful completion of the robotic pick and place task.

Autonomy has additional limitations on the suitability of the resulting robotic system to practical applications. By requiring the intervention of humans in the robotic construction process, the system has limited use cases within the architecture and construction industries in its present state. However, using the metrics of autonomy presented earlier, this system can be evaluated as to the relevance and success of the system in achieving a suitable level of autonomy. Firstly, collecting environment information is manifested in this system autonomously through an environment sensor and requires no human input to function successfully. Secondly, environment perception is addressed in terms of gathering information but not intelligently preempting critical decision-making processes. Thirdly, localisation is addressed through this system by relating collected environment information to the position of the robot to localise it within the operating environment. Fourthly, a minimal amount of rudimentary decision making is undertaken by the system in terms of what object to initiate interaction with first. The system does not make any further attempts to implement complex decision-making processes. Finally, motion instruction execution is dealt with through ROS thereby exhibiting the level of autonomy that is inherent in the system.

The combination of these five behaviours dictates the level of autonomy that an artificial robotic system possesses. Based on the resulting system of this thesis it is clear that the system obtains a moderate level of autonomy, capable of medium complexity tasks, however, it is unable to be classed as fully autonomous due to the requirement for human interaction in several critical stages of the wall construction process.

9.1.4 Decision making

Decision making is a fundamental consideration of an intelligent system and serves to act as a differentiator between simplistic artificial systems and artificial systems that would be considered intelligent. When evaluating the outcomes of this thesis in terms of being able to make decisions, specific abilities as discussed earlier, offer a means of relating decision-making theory to practical outcomes. Decision making can be thought of as having the ability to evaluate the current state of the operating environment and, based on this information, execute instructions on the robot that makes use of this information to inform the actions of the robot. Decision making has been implemented in the final prototype of this thesis by requiring the robot to decide what position a block should be placed in a user-defined block wall structure. Through the use of deep learning algorithms, the robotic system decides what the orientation of each block is and in turn, which pile of blocks to use to construct the correct wall structure. Decision making is a fundamental process in intelligent systems delivering more appropriate approaches to the undertaking of tasks than those artificial systems without decision making.

9.1.5 Learning

Learning and the process of learning have been less widely investigated through the development of this thesis. This is primarily due to the complexity of implementing learning systems in robotic processes that rely on accurate mappings of the operating environment to function correctly. While not actively implemented to the extent required to call the process learning throughout the development of the robotic pick and

place system, elements of the learning process have been implemented to improve the performance of the system. Brooks (2014) suggests that "Most of what people do in their day to day lives is not problem-solving or planning, but rather it is routine activity in a relatively benign, but certainly dynamic, world" (p. 1) and as such pre-programming, the ability to solve specific problems that the system is expected to encounter, can and does constitute a level of self-learning. This level of self-learning has been implemented through the design process to formalise complex underlying systems in terms that are useful to the robot.

9.2 Autonomous vehicles

The role of autonomous vehicles throughout the development of this thesis has been to act as a roadmap/framework for the development of an autonomous robotic construction system. The direction and/or outcomes of each prototype have evolved to follow a similar direction to that of the development of autonomous vehicles over the last ten years. Given the complexity of an autonomous vehicle, the progress of this research has not completed the autonomous vehicle roadmap. However, progress has been made towards translating portions of the framework from autonomous vehicles to robotic pick and place operations. This includes, operating environment visual information gathering and using this information to make decisions about the best course of action for the system to take in order to achieve the required task of constructing a block wall.

9.3 Limitations

Throughout the design process, several limitations became apparent. These limitations primarily related to issues in accurately translating physical operating environment variables into digital terms useful to the robot. These limitations can be broadly categorised into two areas; environment and technical. Both groups of limitations are restrictive to the development of the research. However, environmental limitations can be more easily developed out of the system due to the relative ease in manipulating the physical environment to better suit the

operating conditions of the robot. In contrast, technical limitations require a considerable amount of technical competence to develop comprehensive solutions to primarily high-level programmatic issues.

When individually examining the limitations of the final outcomes of this thesis two fundamental limitations present themselves as critical issues to the overall success of autonomous robotic pick and place operations for architectural construction. Firstly, operating environment setup. Strict limitations are required in order to allow the robot to function successfully in the prescribed operating environment. The objects with which the robot is required to interact must not interfere with each other and external clutter must be removed in order to map the operating environment with enough clarity to undertake construction tasks. Secondly, the limitations of the technical aspects of the robot environment interaction process limit the feasibility of this prototype. This limitation relates to the complexity of differentiating between objects in the operating environment ie.objects that are not in the same plane as the operating environment ground surface. This is a significant limitation that restricts the overall ability of the robotic system to perform complex construction tasks.

9.4 Opportunities

Several opportunities exist around these limitations in which the further development of this research could enhance the ability of the system to perform complex construction tasks. Opportunities primarily exist to develop a more complex operating environment mapping system capable of multi-angle object detection and recognition. This would allow a developed robotic system to grasp a group of blocks without the placement of such blocks being essential to the success of the grasping operation. In addition, greater environment mapping accuracy would allow the robotic system to begin to make more complex decisions about how to best interact with the environment thereby internally evaluating the best course of action to take given the information it has about the operating environment.

Further opportunities also exist in which to continue the development of this research. Firstly, implementing an image-based processing system to

identify the location of blocks within the environment would have two significant benefits; firstly, processing performance would significantly increase due to the ability for computational system to more quickly process the image data; secondly, the implementation of a more complex deep learning algorithm would be advantageous due to the system using an image-based framework over a point cloud framework. The combination of these two opportunities suggests a higher level of task execution complexity could be achieved if additional resources were attributed to the research.

The outcomes of this design-based research have developed a competent robotic system capable of executing pick and place operations in a controlled environment. Additionally, this system is capable of acting with a level of intelligence and autonomy that begins to demonstrate the implications of such a system on the architecture and construction industries. Although several limitations reduce the viability of the system, a significant number of opportunities exist that could develop this research into a fully autonomous construction tool.

10. Conclusion

Robotic fabrication and robotic construction are two essential developments in the construction industry. Both of these construction methodologies provide a means of increasing the rate at which complex architectural structures can be built. The development of autonomous systems within this industry is vital to improving the abilities of robotic systems to undertake essential construction tasks with a better level of competency than human workers. This thesis has developed a robotic system capable of autonomously constructing a simple block wall through the use of real-time feedback, artificial intelligence algorithms and complex decision-making processes.

Essential to the development of the outcomes of this thesis are critical aspects of the literature, which provide a direction to the scope of the research. Firstly, autonomy was identified as a critical contributor to an intelligent system with the level of autonomy contributing directly to the abilities of the robotic system. Secondly, perception, the ability to make sense of the environment, is an equally important descriptor of an intelligent robotic system. Perception allows a robotic system to act responsively to changes in operating environment conditions in a manner that is consistent with existing intelligent systems. Thirdly, decision making contributes to the intelligence of a robotic system by allowing such a system to critically evaluate the state of the operating environment and then act most appropriately in order to achieve a pre-defined goal. Finally, feedback loops provide an intelligent robotic system with a means of understanding changes in the operating environment and adapting future actions to these changes. The combination of these key theoretical elements define the structure of robotic systems intending to operate intelligently in a variety of environments.

The resulting outcomes of this research are two robotic systems with a similar set of capabilities however, one develops through learning while the other is conventionally programmed to interact with the operating environment. These two systems demonstrate different levels of intelligence based on the two different approaches to environment interaction. Prototype Two develops a means of allowing the robot to interact with objects in the operating environment. In contrast Prototype Three resets to conventional

methods of environment interaction however the interface between the operator and the robotic system is defined by the operator providing the robot with an image from which it is to construct a block wall structure.

The potential for this research to form the beginnings of foundational changes to the architecture and construction industries should not be understated. The implications of autonomous robotic systems that can successfully execute complex construction tasks is significant. These systems can operate consistently for extended periods while maintaining a higher level of accuracy than any human construction worker could attain. This has significant implications for the future role of the architect in the construction environment with the potential for the architect to have greater control over the construction process by utilising autonomous robotic systems to manifest complex architectural structures.

11. Bibliography

Baareh, A. K., Sheta, A. F., & Al-batah, M. S. (2012). Feature based 3D Object Recognition using Artificial Neural Networks. International Journal of Computer Applications in Technology, 44(5). https://doi.org/10.5120/6256-8402

Bellman, R. E., & Zadeh, L. A. (1970). Decision-Making in a Fuzzy Environment. Management Science, 17(4), 141–165.

Besl, P. J., & McKay, N. D. (1992). A method for registration of 3-D shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence, 14(2), 239–256.

Binet, A. (1905). New Methods for the Diagnosis of the Intellectual Level of Subnormals. L'Année Psychologique, 191–244.

Blais, G., & Levine, M. D. (1995). Registering multiview range data to create 3D computer objects. IEEE Transactions on Pattern Analysis and Machine Intelligence, 17(8), 820–824.

Brooks, R. A. (1991). Intelligence Without Reason. Massachusetts Institute of Technology. http://www.sciencedirect.com/science/article/pii/000437029190053M

Brooks, R. A. (2014). The Role of Learning in Autonomous Robots. Fourth Annual Workshop on Computation Learning Theory, 5–10.

De Gregorio, D., Tombari, F., & Di Stefano, L. (2016). RobotFusion: Grasping with a Robotic Manipulator via Multi-view Reconstruction. In G. Hua & H. Jégou (Eds.), Computer Vision – ECCV 2016 Workshops. Springer International Publishing.

Eversmann, P. (2018). Robotic Fabrication Techniques for Material of Unknown Geometry. In K. De Rycke, C. Gengnagel, O. Baverel, J. Burry, C. Mueller, M. M. Nguyen, P. Rahm, & M. R. Thomsen (Eds.), Humanizing Digital Reality: Design Modelling Symposium Paris 2017 (pp. 311–322). Springer Singapore.

FlexBrick, ETH Zurich, 2008-2010. (n.d.). Gramazio Kohler Research. Retrieved March 18, 2019, from http://gramaziokohler.arch.ethz.ch/web/e/forschung/152.html

Froese, T., Virgo, N., & Izquierdo, E. (2007). Autonomy: A Review and a Reappraisal. In F. Almeida e Costa, L. M. Rocha, E. Costa, I. Harvey, & A. Coutinho (Eds.), Advances in Artificial Life (Vol. 4648, pp. 455–464). Springer Berlin Heidelberg.

Furrer, F., Wermelinger, M., Yoshida, H., Gramazio, F., Kohler, M., Siegwart, R., & Hutter, M. (2017). Autonomous robotic stone stacking with online next best object target pose planning.

Goodfellow, I., Bengio, Y., & Courvile, A. (2016). Deep Learning. MIT Press.

Guérin, J., Thiery, S., Nyiri, E., & Gibaru, O. (2018). Unsupervised robotic sorting: Towards autonomous decision making robots. In arXiv [cs.RO]. arXiv. http://arxiv.org/abs/1804.04572

Helm, V., Willmann, J., Gramazio, F., & Kohler, M. (2014). In-Situ Robotic Fabrication : Advanced Digital Manufacturing Beyond the Laboratory. In F. Röhrbein, G. Veiga, & C. Natale (Eds.), Gearing up and accelerating cross-fertilization between academic and industrial robotics research in Europe : technology transfer experiments from the ECHORD project (Vol. 94, pp. 63–83). Springer International Publishing.

Lucas, R. (2016). Research methods for architecture. Laurence King Publishing London.

Maes, P. (1990). Situated agents can have goals. Robotics and Autonomous Systems, 6(1), 49–70.

Maes, P. (1993). Modelling Adaptive Autonomous Agents. Artificial Life, 1(1-2), 135–162.

Martín, R. M., Lorbach, M., & Brock, O. (2014). Deterioration of depth measurements due to interference of multiple RGB-D sensors. IEEE.

Nagy, Z. (1994). Object Recognition by Neural Network. IFAC Proceedings Volumes, 27(3), 169–173.

Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohi, P., Shotton, J., Hodges, S., & Fitzgibbon, A. (2011). KinectFusion: Real-time dense surface mapping and tracking. 2011 10th IEEE International Symposium on Mixed and Augmented Reality.

Pfeifer, R. (1996). Building "Fungus Eaters": Design Principles of Autonomous Agents. In Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior SAB96 (From Animals to Animats), 3–12.

Pinto, A. M., Rocha, L. F., & Paulo Moreira, A. (2013). Object recognition using laser range finder and machine learning techniques. Robotics and Computer-Integrated Manufacturing, 29(1), 12–22.

Rahman, M. M., Tan, Y., Xue, J., Shao, L., & Lu, K. (2019). 3D object detection: Learning 3D bounding boxes from scaled down 2D bounding boxes in RGB-D images. Information Sciences, 476, 147–158.

Saaty, T. L. (2008). Decision making with the analytic hierarchy process. Ethical Human Sciences and Services: An International Journal of Critical Inquiry, 1(1), 83–98.

Smithers, T. (1997). Autonomy in robots and other agents. Brain and Cognition, 34(1), 88–106.

Sobti, A., Arora, C., & Balakrishnan, M. (2018). Object Detection in Real-Time Systems: Going Beyond Precision. 1020–1028.

Sorin, D., & Konidaris, G. (2018). Enabling Faster, More Capable Robots With Real-Time Motion Planning. IEEE Spectrum: Technology, Engineering, and Science News; IEEE Spectrum. https://

spectrum.ieee.org/automaton/robotics/robotics-software/enabling-faster-more-capable-robots-with-real-time-motion-planning

Sternberg, R. J., & Salter, W. (1982). Conceptions of Intelligence. In R. J. Sternberg (Ed.), Handbook of Human Intelligence (pp. 3–15). Cambridge University Press.

Stratifications, London, 2011. (n.d.). Gramazio Kohler Research. Retrieved March 18, 2019, from http://gramaziokohler.arch.ethz.ch/web/e/projekte/206.html

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Thondiyath, A. (2016). Autonomy for Robots: Design and Developmental Challenges (Keynote Address). Procedia Technology, 23, 4–6.

Tsarouchi, P., Matthaiakis, S.-A., Michalos, G., Makris, S., & Chryssolouris, G. (2016). A method for detection of randomly placed objects for robotic handling. CIRP Journal of Manufacturing Science and Technology, 14, 20–27.

Wechsler, D. (1944). The measurement of adult intelligence. Williams & Wilkins Co.

Ziegler, J., Gattringer, H., Kaserer, D., & Müller, A. (2018). Automated, Depth Sensor Based Object Detection and Path Planning for Robot-Aided 3D Scanning. Advances in Service and Industrial Robotics, 336–343.

12. List of Figures

13. Appendix

13.1 Robot Control Code

13.1.1 Main Control Program

```
#!/usr/bin/env python

import tf
import sys
import time
import math
import rospy
import capture
import file_io
import requests
from math import radians
from pprint import pprint
from block_wall import blockWall
from object_pose import PosePoint
from pick_object import PickPlace
from req_grip import gripperControl
from camera_pos import camera_capture
from wall_retention import wallRetention
from requests.auth import HTTPDigestAuth
from motion_control import MotionControl
from fail_position import failed_exec_pos


class mainControl():

    def __init__(self):

        wallRetention().write_base()

          url = 'http://192.168.125.1/rw/iosystem/signals/gripper_open?action=set'
        payload = {'lvalue': 1}
        self.r = requests.post(
            url, data=payload, auth=HTTPDigestAuth('Default User', 'robotics'))

    def number_blocks(self):
        file_main = file_io.read_file(
            '/home/harrison/Desktop/Share/point_cloud/no_blocks.csv', 'n')
        if file_main[0] == 'None':
            no_blocks = 0
        else:
            no_blocks = int(file_main[0])

        print '[INFO]: Number of blocks {}.'.format(no_blocks)

        return no_blocks

    def pick_targets(self):
```

```
        # Read in the coordinate values from rhino
        x_values = file_io.read_file(
            '/home/harrison/Desktop/Share/point_cloud/motion_commands.csv', 'x')
        y_values = file_io.read_file(
            '/home/harrison/Desktop/Share/point_cloud/motion_commands.csv', 'y')
        z_values = file_io.read_file(
            '/home/harrison/Desktop/Share/point_cloud/motion_commands.csv', 'z')

        # Split list into each block
        point_list = [[[float(x_values.pop(0)),
                        float(y_values.pop(0)),
                        float(z_values.pop(0))]
                       for _ in range(4)]
                      for _ in range(self.number_blocks())]

        pprint(point_list)
        return point_list

    def point_capture(self):

        # Move to camera capture position
        move = MotionControl([camera_capture()])
        move.main()
        # Wait for the robot to move to the capture position
        time.sleep(1)
        # Execute the pointcloud capture
        capture.main()
        # Wait for processing in rhino
        time.sleep(5)

    def execute_motion(self):

        for i, block in enumerate(self.pick_targets()):
            print '[INFO] Loop {}'.format(i)
            rot_angle = PosePoint(block[0], block[1], block[2])
            print '[INFO]: Rotation Angle: {}'.format(rot_angle.z_angle())
            quaternion_rotation = tf.transformations.quaternion_from_euler(
                rot_angle.z_angle(), radians(0), radians(180))
            print '[INFO]: Quaternion Rotation:'
            pprint(quaternion_rotation)
            centre_point = block[3]
            print '[INFO]: Centre Point: {}'.format(centre_point)
            quaternion_rotation = list(quaternion_rotation)
            for item in quaternion_rotation:
                centre_point.append(item)
            print '[INFO]: Motion Target: {}'.format(centre_point)

            print '[INFO]: Executing pick operation'
            pick_object = PickPlace('pick', [centre_point], 75, self.r)
            pick_object.main()
```

```
            print '[INFO]: Executing place operation'
            place_object = PickPlace(
                'place', [wallRetention().write_file()], 75, self.r)
            place_object.main()

    def main(self):

        while len(wallRetention().readin_file()) > 0:
            self.point_capture()
            while self.number_blocks() == 0:
                time.sleep(8)
                self.main()
            self.execute_motion()
            self.main()


if __name__ == "__main__":

    try:
        motion = mainControl()
        motion.main()

    except rospy.ROSInterruptException:
        pass
```

13.1.2 Wall Generation Program

```
#!/usr/bin/env python

import tf
from pprint import pprint
from math import degrees, radians


class blockWall(object):

    '''
    Class to generate the coordinate positions of a simple linear block wall
    Usage: <block_length>(block length in mm),
           <block_width>(block width in mm),
           <block_height>(block height in mm),
           <wall_length>(number of blocks in a row),
           <wall_height>(number of rows),
           <start_point>([x, y, z]),
           <wall_direction>(x, -x, y, -y),
           <mortar_spacing>(spacing between bricks in mm)
           <row_offset>(offset between alternate rows in mm)
    '''

    def __init__(self, block_length, block_width, block_height, wall_length, wall_height,
start_point, wall_direction, mortar_spacing, row_offset):
        self.block_length = block_length
        self.block_width = block_width
        self.block_height = block_height
        self.wall_length = wall_length
        self.wall_height = wall_height
        self.start_point = start_point
        self.wall_direction = wall_direction
        self.mortar_spacing = mortar_spacing
        self.row_offset = row_offset

    def height_control(self):
        '''
        Function to generate the robot targets for the specified number of rows
        '''

        place_list = []

        for i in range(self.wall_height):
            increment = self.block_height * i
            if i % 2 == 0:
                   place_list.append(self.row_control('norm', self.start_point[2] + (self.
block_height + increment)))
            if i % 2 != 0:
                    place_list.append(self.row_control('alt', self.start_point[2] + (self.
block_height + increment)))
        return place_list
```

```
    def row_control(self, norm_alt, plane_height):
        '''
        Function to generate block placement coordinates for inputs
        function allows for the specification of the direction in which the wall is
        generated from the start point
        '''

        mortar_spacing = self.mortar_spacing
         # The rotation angle of the blocks for a wall to be constructed perpendicular to
the robot
           x_wall_angle = tf.transformations.quaternion_from_euler(radians(0), radians(0),
radians(180))
        x_wall_angle = list(x_wall_angle)
       # The rotation angle of the blocks for a wall to be constructed parallel to the robot
          y_wall_angle = tf.transformations.quaternion_from_euler(radians(90), radians(0),
radians(180))
        y_wall_angle = list(y_wall_angle)
        # Start point coordinates
        x_coordinate = self.start_point[0]
        y_coordinate = self.start_point[1]

        point_list = []

        if norm_alt == 'norm':

            # Test to determine the direction of the wall based on user input
            if self.wall_direction == '-x' or self.wall_direction == 'x':
                # Loop for the number of blocks in each row
                for i in range(1, self.wall_length + 1):

                    # Empty point list
                    point = []

                    # Test which direction the wall is going in
                    if self.wall_direction == '-x':
                      point.append(self.start_point[0] - (self.block_length * i) - (mortar_
spacing * i))
                        point.append(y_coordinate)
                    if self.wall_direction == 'x':
                            point.append(self.start_point[0] + (self.block_length * i) +
(mortar_spacing * i))
                        point. append(y_coordinate)

                    # Append the height of the block as the z coord place value
                    point.append(plane_height)

                    point_list.append(point)

                # Add the quaternion description to the end of each point in the list
                for item in point_list:
                    for angle in x_wall_angle:
```

```
                        item.append(angle)

                return point_list

            # Test to determine the direction of the wall based on user input
            if self.wall_direction == '-y' or self.wall_direction == 'y':

                # Loop for the number of blocks in each row
                for i in range(1, self.wall_length + 1):

                    # Empty point list
                    point = []

                    # Test which direction the wall is going in
                    if self.wall_direction == '-y':
                        point.append(x_coordinate)
                      point.append(self.start_point[1] - (self.block_length * i) - (mortar_
spacing * i))
                    if self.wall_direction == 'y':
                        point.append(x_coordinate)
                             point.append(self.start_point[1] + (self.block_length * i) +
(mortar_spacing * i))

                    # Append the height of the block as z coord place value
                    point.append(plane_height)

                    point_list.append(point)

                # Add th quaternion description to the end of each point in the list
                for item in point_list:
                    for angle in y_wall_angle:
                        item.append(angle)

                return point_list

        if norm_alt == 'alt':

            # Test to determine the direction of the wall based on user input
            if self.wall_direction == '-x' or self.wall_direction == 'x':
                # Loop for the number of blocks in each row
                for i in range(1, self.wall_length + 1):

                    # Empty point list
                    point = []

                    # Test which direction the wall is going in
                    if self.wall_direction == '-x':
                          point.append((self.start_point[0]) - (((self.block_length * i) -
(mortar_spacing * i)) - (self.row_offset)))
                        point.append(y_coordinate)
                    if self.wall_direction == 'x':
                          point.append(((self.start_point[0]) + ((self.block_length * i) +
```

```
(mortar_spacing * i)) + (self.row_offset)))
                        point. append(y_coordinate)

                    # Append the height of the block as the z coord place value
                    point.append(plane_height)

                    point_list.append(point)

                # Add the quaternion description to the end of each point in the list
                for item in point_list:
                    for angle in x_wall_angle:
                        item.append(angle)

                return point_list

            # Test to determine the direction of the wall based on user input
            if self.wall_direction == '-y' or self.wall_direction == 'y':

                # Loop for the number of blocks in each row
                for i in range(1, self.wall_length + 1):

                    # Empty point list
                    point = []

                    # Test which direction the wall is going in
                    if self.wall_direction == '-y':
                        point.append(x_coordinate)
                          point.append(((self.start_point[1]) - ((self.block_length * i) -
(mortar_spacing * i)) - (self.row_offset)))
                    if self.wall_direction == 'y':
                        point.append(x_coordinate)
                          point.append(((self.start_point[1]) + ((self.block_length * i) +
(mortar_spacing * i)) + (self.row_offset)))

                    # Append the height of the block as z coord place value
                    point.append(plane_height)

                    point_list.append(point)

                # Add the quaternion description to the end of each point in the list
                for item in point_list:
                    for angle in y_wall_angle:
                        item.append(angle)

                return point_list
```

13.1.3 Motion Instructions

```
#!/usr/bin/env python

import tf
import sys
import time
import rospy
import irb1200_home
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
from math import radians, degrees


class MotionControl(object):

    def __init__(self, target_list):

        self.target_list = target_list

        moveit_commander.roscpp_initialize(sys.argv)
        rospy.init_node('irb1200_motion_commands', anonymous=True)

        self.robot = moveit_commander.RobotCommander()
        self.scene = moveit_commander.PlanningSceneInterface()
        self.move_group = moveit_commander.MoveGroupCommander('irb1200_arm')
        self.display_trajectory_publisher = rospy.Publisher(
             '/move_group/display_planned_path', moveit_msgs.msg.DisplayTrajectory, queue_
size=20)

    def start_robot(self):

        # Print for debugging
        print 'Reference frame: {}'.format(
            self.move_group.get_planning_frame())
        print self.move_group.get_end_effector_link()
        print 'Robot Groups'
        print self.robot.get_group_names()
        print 'Printing robot state'
        print self.robot.get_current_state()
        print ''

    def motion_points(self):

        # Generate a grid of points
        test2 = tf.transformations.euler_from_quaternion(
            [0.00, 0.707106, 0.0, -0.707106])

        test2 = list(test2)

        z_angle = radians(-180.0 + 38.660827)
```

```
        y_angle = radians(-90.0)
        x_angle = radians(-10.0)

        test_point = tf.transformations.quaternion_from_euler(
            x_angle, y_angle, z_angle)

        print '{}\n'.format(self.target_list)

        print 'Executing motion . . .'

        # For each point in the grid move to that point
        for target in self.target_list:
            pose_target = geometry_msgs.msg.Pose()
            pose_target.orientation.w = target[3]
            # pick_pose[0]  # 0.707106781187
            pose_target.orientation.x = target[4]
            pose_target.orientation.y = target[5]  # pick_pose[1]  # 0.0000
            # 0.0000  # -0.707106781187
            pose_target.orientation.z = target[6]  # pick_pose[2]
            pose_target.position.x = float(round(target[0] * 0.001, 4))
            pose_target.position.y = float(round(target[1] * 0.001, 4))
            pose_target.position.z = float(round((target[2]) * 0.001, 4))
            self.move_group.set_pose_target(pose_target)
            self.move_group.go(wait=True)

            # If motion failed go to home position
            # and end motion execution
            if self.move_group.go(wait=True) == False:
                home = irb1200_home.abb1200MoveGroupInteface()
                home.go_to_joint_state()
                break

            print 'Moved to target {}'.format(target)
            print '{}\n'.format(self.move_group.get_current_pose())
            self.move_group.clear_pose_targets()

    def main(self):

        self.start_robot()
        self.motion_points()
```

13.1.4 Grasping Instructions

```
#!/usr/bin/env python

import time
from req_grip import gripperControl
from pick_motion import MotionControl


class PickPlace(object):

    def __init__(self, pick_place, grab_release_point, offset_point, session):

        self.pick_place = pick_place
        self.approach_point = [[grab_release_point[0][0],
                                    grab_release_point[0][1], grab_release_point[0][2] +
offset_point,
                                grab_release_point[0][3], grab_release_point[0][4],
                                grab_release_point[0][5], grab_release_point[0][6]]]

        self.grab_release_point = grab_release_point
        self.withdraw_point = self.approach_point
        self.session = session

    def approach(self):
        '''
        Function executed when approaching a pick / place target
        '''

        # if executing a pick operation
        if self.pick_place == 'pick':
            # Open the grippers
            gripperControl().open_close('open', self.session)
            # Wait for 2 seconds to ensure grippers are fully open
            time.sleep(0.25)

        # if executing a place operation do nothing
        elif self.pick_place == 'place':
            pass

        # Move to the approach point
        motion = MotionControl(self.approach_point)
        motion.main()

    def grab_release(self):
        '''
        Function executed when either picking or placing an object
        '''

        # Move to the pick / place target
        motion = MotionControl(self.grab_release_point)
        motion.main()
```

```
        # if executing a pick operation
        if self.pick_place == 'pick':
            # Close the grippers
            gripperControl().open_close('close', self.session)
            # Wait for 2 seconds to ensure object is properly grasped
            time.sleep(0.25)

        # if executing a place operation
        elif self.pick_place == 'place':
            # Open the grippers
            gripperControl().open_close('open', self.session)
            # Wait for 2 seconds to ensure grippers are fully open
            time.sleep(0.25)

    def withdraw(self):
        '''
        Function executed when withdrawing from a pick / place target
        '''

        # Move to the withdraw target
        motion = MotionControl(self.withdraw_point)
        motion.main()

    def main(self):
        '''
        Execute the pick and place operation
        '''

        self.approach()
        self.grab_release()
        self.withdraw()
```

13.1.5 Block Rotation Angle

```
#!/usr/bin/env python

from sympy import Point3D, Line
from math import degrees, radians


class PosePoint(object):

    '''
    Generates the centre point and z axis rotation for an orthogonal object defined by three
points.
    Todo:
    Add functionality to calculate x and y axis rotation
    Usage: PosePoint(<origin[x, y, z]>, <point_1[x, y, z]>, <point_2[x, y, z]>)
    '''

    def __init__(self, point_1, point_2, point_3):
        self.point_1 = point_1
        self.point_2 = point_2
        self.point_3 = point_3

    def pose_point(self, point):
        '''
        Function to seperate a point into x, y and z components.
        '''

        x = float(point[0])
        y = float(point[1])
        z = float(point[2])

        return x, y, z

    def plane_points(self):
        '''
        Convert entered floating point values into native 3D point values
        for future use
        '''

        origin = Point3D(self.pose_point(self.point_1))
        max_y = Point3D(self.pose_point(self.point_2))
        min_y = Point3D(self.pose_point(self.point_3))

        return origin, max_y, min_y

    def normal_vector(self):
        '''
        Generate two vectors which describe the short and long sides of an orthogonal
        block
        '''
```

```
        # Get the three points required to define a plane
        origin, max_y, min_y = self.plane_points()

        # Calculate the distance between the origin and the first point
        length_one = origin.distance(max_y)

        # Calculate the distance between the origin and the second point
        length_two = origin.distance(min_y)

        # Print the lengths to the terminal for debugging
        print 'Length One: {}'.format(float(length_one))
        print 'Length Two: {}'.format(float(length_two))

        # Seperate the origin into coordinate components
        x_point = origin.x
        y_point = origin.y
        z_point = origin.z

        # Offset y component to calculate a normal vector for angle calculation
        new_x_point = y_point - 100

        # Generate a new point
        test_point = Point3D(x_point, new_x_point, z_point)

        # Generate a new base line for angle measurement
        line_1 = Line(origin, test_point)

        return line_1, length_one, length_two

    def z_angle(self):

        # Get the three required points
        origin, max_y, min_y = self.plane_points()

        # Get the three required vectors
        normal_vector, length_one, length_two = self.normal_vector()

        # Test to determine which vector is the shortest
        # This is used to determine which side of the block is the shortest
        if length_one > length_two:

            line_2 = Line(origin, min_y)

            # Calculate the angle between the base line and the shortest
            # side of the block
            line_angle = normal_vector.angle_between(line_2)

            if degrees(line_angle) <= 5:

                line_angle = 0

            # Return the required rotation angle for the robot to align
```

```
            # correctly with the block
            print degrees(float(-line_angle))
            # return float(-line_angle - radians(90))
            return float(-line_angle)

        elif length_one < length_two:

            line_2 = Line(origin, max_y)

            # Calculate the angle between the base line and the shortest
            # side of the block
            line_angle = normal_vector.angle_between(line_2)

            if degrees(line_angle) <= 5:

                line_angle = 0

            # Return the required z axis rotation for the robot to align
            # correctly with the block
            print degrees(float(-line_angle))
            # return float(-line_angle - radians(90))
            return float(-line_angle)

    def centre_point(self):
        '''
        Calculates the centre point for a block defined by three points
        '''

        # Get the three required points
        origin, max_y, min_y = self.plane_points()

        # Caculate the centre point
        centre_point = [(float((max_y.x + min_y.x) / 2)),
                        (float((max_y.y + min_y.y) / 2)), float(max_y.z)]

        return centre_point
```

13.1.6 Capture Pointcloud

```
Learn more or give us feedback
#!/usr/bin/env python

import csv
import time
import rospy
from sensor_msgs.point_cloud2 import read_points_list, PointCloud2


class camCap:

  def __init__(self):
      self.sub_once = rospy.Subscriber("/camera/depth/color/points", PointCloud2, self.
callback)

  def callback(self, data):

    start_time = time.time()
    print '[INFO]: Starting point cloud capture at...'
    data_list = read_points_list(data)
   with open('/home/harrison/Desktop/Share/point_cloud/final_points.csv', 'w') as new_file:
        csv_writer = csv.writer(new_file, delimiter=',')

        csv_writer.writerow('xyzc')

        for line in data_list:
            csv_writer.writerow(line)

    end_time = time.time()
    print '[INFO]: Data extraction complete at...'
    print '[INFO]: Processing complete in {}'.format(end_time - start_time)

    self.sub_once.unregister()


def main():

    rospy.init_node('irb1200_motion_commands', anonymous=True)
    camCap()
```

13.1.7 Wall Retention

```
#!/usr/bin/env python

import file_io
import random
from pprint import pprint
from block_wall import blockWall

class wallRetention:

    def write_base(self):

        wall_selection = random.randint(1, 3)
        print wall_selection

        if wall_selection == 1:

            wall_position = [250, 280, 2]
            wall_direction = 'x'

        elif wall_selection == 2:

            wall_position = [250, -280, 2]
            wall_direction = 'x'

        elif wall_selection == 3:

            wall_position = [600, -140, 2]
            wall_direction = 'y'

        # Define wall variables
        mortar_spacing = 10
        alt_row_offset = 20
        no_rows = int(20/4)

        # Generate the wall robot targets
        place_targets = blockWall(70, 30, 30, 4, no_rows, wall_position,
                                  wall_direction, mortar_spacing,
                                  alt_row_offset)

        random_list = []

        for item in place_targets.height_control():
            for coord in item:
                random_list.append(coord)

        # Write the file to shared directory
       file_io.save_file('/home/harrison/Desktop/Share/point_cloud/base_wall.csv', random_
list)

        print '[INFO]: Wall block coordinates generated'
```

```
    def readin_file(self):

        # Readin the target coordinate file
        test_file = file_io.read_line('/home/harrison/Desktop/Share/point_cloud/base_wall.
csv')

        # Print for debugging
        pprint(test_file)

        return test_file

    def write_file(self):

        # Read in the wall target list
        target = self.readin_file()
        actual_target = target.pop(0)

        final_targets = []

        for item in actual_target:
            # Convert the targets to floating point values
            final_targets.append(float(item))

        # Save the file with the used target removed
       file_io.save_file('/home/harrison/Desktop/Share/point_cloud/base_wall.csv', target)

        print '[INFO]: Target extracted & file written'
        # Return the target for robot execution
        return final_targets
```

13.1.8 Request Tool Action

```
#!/usr/bin/env python

import time
import requests
from requests.auth import HTTPDigestAuth


class gripperControl:


    def gripper_url(self, open_close, value, s):

        if open_close == 'open':

            url = 'http://192.168.125.1/rw/iosystem/signals/gripper_open?action=set'
            payload = {'lvalue': value}
                r = requests.post(url, data=payload, auth=HTTPDigestAuth('Default User',
'robotics'), cookies=s.cookies)
            print '[INFO]: Status {}'.format(r.status_code)
            print '[INFO]: Text {}'.format(r.text)
            print '[INFO]: Cookies {}'.format(r.cookies)

        elif open_close == 'close':

            url = 'http://192.168.125.1/rw/iosystem/signals/gripper_close?action=set'
            payload = {'lvalue': value}
                r = requests.post(url, data=payload, auth=HTTPDigestAuth('Default User',
'robotics'), cookies=s.cookies)
            print '[INFO]: Status {}'.format(r.status_code)
            print '[INFO]: Text {}'.format(r.text)
            print '[INFO]: Cookies {}'.format(r.cookies)

    def open_close(self, open_close, session):

        if open_close == 'open':
            self.gripper_url('open', 0, session)
            self.gripper_url('close', 0, session)
            self.gripper_url('open', 1, session)
            time.sleep(0.5)

        elif open_close == 'close':
            self.gripper_url('open', 0, session)
            self.gripper_url('close', 0, session)
            self.gripper_url('close', 1, session)
            time.sleep(0.5)
```