



School of Electronic Engineering and Computer  
Science

Final Year Undergraduate Project 2017/2018

Final Report

BEng Electrical and Electronic Engineering

# Touch it, move it, feel it: Enhanced Object Understanding Through Robot Actions

*Student Name:*

Nnadozie Okeke

*Supervisor:*

Dr Lorenzo Jamone

26 April 2018

## Abstract

This project set out to answer the question: In learning affordances, does tactile data combined with visual data improve the performance of a robotic system in detecting tool dependent affordances? Affordances of an object are the action possibilities the object permits an observer. They are dependent on the object's properties, such as shape, composition, animation and position relative to other objects (1). When an observer can perceive these properties, the affordances of an object can be detected. To truly automate the tasks humans do, robots and other assistive systems will need to use the tools humans use. As such they will need to understand the affordances of human tools and more importantly the tool dependent affordances of objects in the human environment. This project presents Sen, which stands for simulation environment. Sen provides two things. First of which is an environment to validate that software used in gathering visual and haptic data by using robotic components will be non-damaging to the real physical robotic hardware. Second of which is a straightforward way of gathering simulated haptic and visual data which can be used in exploring methods for learning affordances. The results from testing Sen suggest that visual and haptic data gathered from a simulation environment can be used in training a Convolutional Neural Network to detect affordances.

## Acknowledgements

I am supremely thankful to:

- my parents for their steadfast support, continuous guidance, gentle motivation, and endless love.
- my supervisor Dr Lorenzo Jamone for giving me this opportunity by accepting my request to carry out this project, as well as his PhD students Gokhan Solak and Sami Siddiqui for their guidance.
- my friends, especially Eva Sf and Oskar Zieba, for their tireless motivation in the final stages of this project.



# Contents

Contents .....	5
Introduction .....	8
1.1 Background to the Subject .....	8
1.2 Motivation .....	10
1.3 Aims and Objectives .....	11
Useful Concepts.....	13
2.1 Convolutional Neural Networks.....	13
2.2 Image Classification.....	13
2.3 Kinematics.....	14
2.4 Object Recognition .....	15
2.4.1 Position Measurement.....	16
2.4.2 Object Detection.....	17
2.4.3 Scene Categorization .....	18
Procedure and Implementation.....	19
3.1 Design Overview.....	19
3.1.1 Consideration of Hardware Components .....	21
3.1.2 Consideration of Software Libraries and Frameworks .....	22
3.1.3 Description of Chosen Components and Software .....	23
3.2 Chronological Order of Implementation.....	26

3.2.1 Choosing an Operating System .....	26
3.2.2 Setting up UR5 Simulation in Gazebo .....	26
3.2.3 Fitting the UR5 with the Allegro Hand.....	27
3.2.5 Challenges Faced in Fixing the Allegro Simulation.....	27
3.2.6 Choosing to Work in VM for Speed.....	28
3.2.7 Real World Camera System .....	29
3.2.9 Creating a UR5 Controller .....	31
3.2.10 Building a Position Publisher Node.....	31
3.2.11 Including a Camera in Gazebo.....	32
3.2.12 Upgrading Packages to Kinetic.....	33
3.2.13 Successfully Streaming Simulated Camera Images.....	33
3.2.14 Publishing End-Effector Positions with Python .....	33
3.2.15 Research into Gathering Force and Contact Data from Gazebo .....	34
Results and Contribution .....	35
5.1 Image Classification Results .....	35
5.2 Stored Image Files and CNN Output Log File.....	37
5.3 Testing the Python Control Interface .....	38
5.4 Error Messages.....	43
5.4.1 Error When Sen Runs as Expected .....	43
5.4.2 Segmentation Fault.....	44
5.4.3 Errors When Collisions Occur .....	45
5.5 Memory Usage.....	46
5.6 Findings of Tactile Sensor Research.....	47
5.7 Contribution .....	47
Discussion .....	49
6.1 Was the aim Achieved? .....	49
6.2 Were Basic Objectives Achieved? .....	50

6.3 Potential for Real World Application .....	51
6.4 Errors Encountered when Running Sen .....	51
Conclusion.....	53
Further Work .....	55
Readme .....	56
Risk Assessment .....	61
A.1 Risks in Academic and Commercial Projects .....	61
A.1.1 Event Tree Analysis .....	63
A.1.2 Fault Tree Analysis.....	64
A.2 Risk Register .....	65
References .....	69

# Chapter 1

## Introduction

### 1.1 Background to the Subject

Enhanced object understanding through robot action is about using robots to improve the understanding of objects through autonomous interactions which gather data that help build more complete understanding of objects. Let us describe robot actions which enhance object understanding as enhancing robot actions. As a quick way to grasp the nature of enhancing robot actions, consider a sorting line in a recycling plant which consists of a conveyor belt on which are placed a mix of materials to be sorted. The sorting line also includes robotic arms positioned along this belt in such a way that they can remove unwanted entities, say blocks of stone, from the conveyor line. Say also that the only sensors available for use in distinguishing wanted entities from unwanted entities are RGB-D cameras positioned above the sorting line and tactile sensors fitted to the robotic arms. If for some reason the images from the RGB-D cameras are not sufficient to classify an object as wanted or unwanted, then an enhancing robot action would be the action involved in using the robotic arms to gather tactile data which helps in correctly classifying the said object. Though contestable, for these actions to be truly robotic they should be autonomous actions performed by robotic systems. So, in our example the robotic arms will autonomously perform the task of gathering more data after registering that the existing data is insufficient for understanding the object.

A major method by which objects are understood is through a perception of their affordances. Affordances of an object are the action possibilities the object permits an observer. They are dependent on the object's properties, such as shape, composition, animation and position relative to other objects (1). When an observer can perceive these properties, the affordances of an object can be detected. For example, if a human adult has



never seen a new cup design before, when the shape of the cup and size relative to the observer are perceived, its action possibilities are quite evident. From the cup's hollow shape, the material it is made of, its relatively small size and its provision for grasping, it will be evident that the cup can be held in a hand and filled with liquid.

With recent advancements in technology enabling better computer vision, image recognition and data gathering systems, there has been a lot of work to apply the concept of affordances in robotics. (1) and (2) are surveys which detail the most prominent endeavors by roboticists to apply the concept of affordances. Most of these have focused on detecting affordances solely through visual perception of objects' properties. However, some have involved the use of depth estimation (3) and acoustic feedback (4). Of particular interest are those works which have sought to detect affordances using haptic feedback. (5) explores haptic detection of affordance using a multi-fingered robot hand which unscrews a bottle screw cap, achieving successful categorization of rotational constraints (screw) and confirming that such constraints can be detected and categorized using haptic feedback. In this case, the haptic feedback was gathered through proprioception. (6) and (7) though not applications of affordances in robotics, explore the ability of humans to detect affordances through haptic means. Specifically, the affordance of a slope for standing. (6) concludes that though slower, haptic feedback is just as accurate as visual judgement in detecting the standing affordance of a slope. Also of interest are works which focus on learning affordances. (8) presents a general model for learning object affordances using Bayesian networks which can deal with uncertainty, redundancy and irrelevant information. It demonstrates successful learning of affordances in the real world by a humanoid robot. (9) and (10) focus on self-supervised learning of affordances. That is learning without human input such as labeled training data. (9) shows that by using co-occurrence of correlated data across separate modalities over time, it is possible to develop a classifier which does not require labels during training. (10) demonstrates that correlated data, such as haptic and depth cues, can be used to learn traverse-ability affordances of objects present in an environment. (11) presents a real-time method to detect affordances from RGB-D images using a Convolutional Neural Network. It demonstrates that these detected affordances can be successfully used by a robot (WALK-MAN) to perform grasp actions. Of great interest are those works which focus on the affordances of tools, an emerging focus in the field of affordances. (12) introduces the computational formalization of affordances into Object Action Complexes (OACs) when exploring how a robot chooses the tools needed to

accomplish a task after acquiring a knowledge of affordances through a combination of vision, learning and control. (13) and (14) investigate detecting affordances of tools. (13) achieves state-of-the-art performance in detecting graspable regions of tools after affordances have been learned from local shape and geometry primitives. Lastly, (14) explores the relative contribution of vision and dynamic touch in perceiving the suitability of tools for certain tasks. Concluding that visual information supersedes touch when both are present, while when limited to touch, suitability ratings are highly dependent on the inertial properties of the tools, and the inertial properties that are needed for a task.

In this project we present Sen, which stands for Simulation environment. Fully complete, it comprises of an RGB-D camera system as well as a robotic arm equipped with a multi-digit anthropomorphic robotic hand that is fitted with tactile sensors. Most importantly Sen provides an environment to validate that software used in gathering visual and haptic data on the robotic components present in the environment will be non-damaging to the real physical hardware. Sen also provides a straightforward way of gathering simulated haptic and visual data which can be used in exploring methods for learning affordances.

## 1.2 Motivation

The field of affordance detection is a maturing field with much to discover. Sit-ability, grasp-ability, open-ability, close-ability, and traverse-ability are common affordances which have been addressed with some success. Despite this success there has been limited exploration of tool dependent affordance detection and a limited scope of sensory data used to detect these affordances, typically the use of only visual sensory data.

As assistive technologies such as virtual home assistants, self-driving cars and humanoid robots become widespread, there is need for affordance aware autonomous systems which can be controlled using very high-level instructions<sup>1</sup>. In a world hungry for improved productivity the future lies in automation of these systems, but the standard approach to automation where spaces are designed from the ground up with automation in

---

<sup>1</sup> High-level instructions refer to instructions given in conversational human language

mind will not work on a truly globally impactful scale. Solely because the world has been designed for humans and cannot be rebuilt for robots. To truly automate the tasks humans do, robots and other assistive systems will need to use the tools humans use. As such they will need to understand the affordances of these tools and very importantly, the affordances that become possible through tool use.

## 1.3 Aims and Objectives

The aim of this project is to answer the question:

In learning affordances, does tactile data combined with visual data improve the performance of a robotic system in detecting tool dependent affordances?

By tool dependent affordances, it is meant action possibilities of an object that cannot be realized without the use of a tool. For instance, in the case of a nail handled by a human hand, hammering of this nail through blocks of wood to hold the blocks together. The aim is to explore the performance benefits, if any, there would be for a system that uses both visual and tactile sensory data in affordance detection, rather than just visual or just tactile.

The objectives of the project were set out as follows:

- Devise a representation of tactile sensory data that would be suitable for use in training convolutional neural networks (CNNs) to detect tool dependent affordances.
- Use both tactile and visual sensory data in training a CNN to detect tool dependent affordances.
- Explore how the loss or addition of the tactile data affects the performance of affordance detection.

In achieving these objectives, the following were needed:

- Tactile and visual data for training and testing a CNN.
- A method for acquiring this data using a robotic arm, a multi-digit anthropomorphic robotic hand, an RGB-D camera, and tactile sensors.
- A simulation environment to test software developed for use on the tools listed above in acquiring this data.

# Chapter 2

## Useful Concepts

In this section we introduce useful concepts which are referred to within this report. Concepts such as Convolutional Neural Networks, image classification, and kinematics. Also, though not explicitly referred to in this report a brief introduction to object recognition as it is applicable to a project of this nature is also included in this section.

### 2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are the current state-of-the-art model architecture for image classification tasks (15). They are very similar to conventional Neural Networks which are inspired by the structure of the brain in that they are made up of neurons. Unlike conventional Neural Networks CNNs explicitly assume that inputs are images. Their forte is classification of images, such as the classification of a hand-written number into one of the ten Arabic numerals 0 –9. In depth treatments of CNNs are provided in (15) and (16).

### 2.2 Image Classification

Image classification refers to the task of mapping an image to a class or grouping similar images into the same class according to information extracted from their visual content. There are two major types of image classification: unsupervised and supervised.

Unsupervised classification is where the outcomes: the groupings of images with common characteristics into the same class, are based on an analysis of the images that is performed without a user providing sample groupings. A user can specify the algorithms used to perform groupings and the desired number of classes but otherwise there is no further input from the user.

Supervised classification on the other hand involves a user providing sample groupings which contain images that have already been mapped to their correct classes. This user provided input is called training data. In situations where there is training data the task is usually for a CNN to come up with a model that appropriately maps the images to the provided classes, rather than to classify the images into groups that contain similar visual characteristics.

Further introduction is provided in (17) which includes instructions on how to implement image classification using leading frameworks such as OpenCV and Caffe.

## 2.3 Kinematics

Kinematics is the relationships between the positions, velocities, and accelerations of the links of a manipulator, where a manipulator is an arm, finger, or leg (18). The application of kinematics analysis allows for control of these manipulators, such as moving their joints into the positions that would in turn allow for an end-effector to move into a desired position. An end-effector is simply the device at the end of a manipulator. In applying Kinematics analysis mathematical formalizations of the relationships specified earlier are used to solve two main problems: Forward kinematics and Inverse Kinematics.

Forward Kinematics involves computing the orientation, velocity and position of an end-effector when the displacements and joint angles of a manipulator are known.

Inverse Kinematics does the inverse, computing the joint parameters that result in an end-effectors position and velocity. An important technique for doing this is the Jacobian inverse technique which is covered in (19) (20) (21) and (22).

A more complete introduction to Kinematics is also provided in (18) and (19).

## 2.4 Object Recognition

Object recognition falls under the domains of computer vision and image processing. Simply put, object recognition describes a process where given some prior knowledge about the appearance of certain objects (a model database created in advance), one or more images are examined to evaluate which objects are present and where (23). A special application which occurs frequently is when the model database contains just one object class, and so the object recognition task boils down to deciding whether an instance of this specific object is present, and where it is. When the task is an industrial one, the term machine vision is usually used in place of object recognition. There are several applications of object recognition, each of which has specific requirements and constraints that have led to a rich diversity of object recognition algorithms. Amongst these are:

- *Position measurement*: where the accurate location of an object is needed. For example, locating the position of ICs (integrated circuits) before they are placed on a PCB (printed circuit board). The  $[x, y]$ -position of the object together with its rotation and scale is often referred to as the object *pose*.
- *Inspection*: where machine vision is applied in a production environment for quality control purposes. This relies on object recognition for locating the part to be inspected, such as the welds or threads of screws.
- *Sorting*: for example, when parcels are sorted depending on size in warehouses. This relies on prior identification and localization of the individual parcels.
- *Counting*: where the number of occurrences of a specific object in an image is required.
- *Object detection*: where an image containing the objects being looked for is compared to a model database with information about the objects, and the process returns a positive if the object exists in the image. This relies on prior training to build up a model of each object.
- *Scene categorization*: Here the aim is not to detect an object in an image but to identify which object class it belongs to. That is, is it a tree, a car, a building, etc? Categorization is therefore classification which matches a semantic meaning to the image.

- *Image retrieval*: here, based on query image containing a specific object, an image database is searched for all images showing the same or similar objects of the same object class.

Sen's capabilities can be enhanced using object detection – to detect from RGB-D images if objects are present in the environment, scene categorization – to classify the objects, and position measurement – to determine the position of object(s) and help guide Sen's robotic arm and hand. Therefore, further treatment of object recognition is limited to only these three applications. An in-depth introduction to object recognition is provided in (23) while (24) (25) (26) are worth looking at for practical work on object recognition.

## 2.4.1 Position Measurement

Recall that the aim of position measurement is to accurately determine the *pose* –  $x$ ,  $y$  position, rotation and scale – of an object in an image. The actual RGB-D camera which is simulated in Sen is the Kinect sensor. The Kinect simultaneously captures depth and colour images that can be used to accurately determine the location of objects in an image. As such in this project there is no concern for implementing position measurement algorithms. However, a brief overview of how the Kinect's position measurement works is provided here, and a more thorough treatment is covered in (27).

The Kinect consists of an infrared emitter, an infrared camera and an RGB camera. The infrared camera and sensor are used to work out how far an object is from the Kinect – which is taken to be the point of origin, while the RGB-D camera captures colour images of the given field of view. To determine the distance of an object the infrared laser source is used to emit a pattern of speckles that are projected onto the scene containing the object. This pattern is captured by the infrared camera and compared to a reference pattern that is obtained by capturing a plane at a known distance from the Kinect. Using the disparity between the captured pattern and the reference pattern, simple triangulation equations are used to work out the distance of objects in the scene. That distance can then be used to calculate the cartesian position of the object with reference to the Kinect.



## 2.4.2 Object Detection

In object detection, an image is checked to see if it contains the object being looked for. On a simpler level however, object detection can refer to checking whether there are any objects in an image without looking to see if the image contains a particular object. Already, well developed 2D/3D vision libraries exist which implement this functionality (28), such as PCL (Point Cloud Library) and OpenCV. As such this section only introduces a popular method used for object detection and gives an overview of PCL libraries which can be used for object detection, as it is possible to obtain not just RGB-D images from the Kinect but also point clouds of a scene.

In image analysis segmentation refers to the process of extracting interesting regions from an image; typically objects that are in the image, as well as boundaries that exist in the image. Usually each pixel in the image is assigned a label such that pixels with identical labels have some characteristics in common. Thus, segmentation can be used to identify what objects are of interest in an image.

PCL provides a segmentation library (29) which contains algorithms for segmenting point clouds into distinct clusters. The theory and implementation of these algorithms are provided in PCL's documentation (30). They include:

- Plane model segmentation,
- Cylinder model segmentation,
- Euclidean Cluster Extraction,
- Region growing segmentation,
- Color-based region growing segmentation,
- Min-cut Based Segmentation, etc.

which should suffice for the purposes of detecting the presence of an object in an image. Combined with PCL's sample consensus library (31), it should be possible to detect when there

is an object in an image, as well as the object's borders. A simple description of the steps involved can be found on ROS answers page (32).

### 2.4.3 Scene Categorization

Here the aim is to identify which class an image belongs to, typically in terms of a semantic label (e.g. office, lake, or building) (33). It is believed that the key to this is having a visual descriptor which describe foundational characteristics such as motion, texture, colour, and shape, to mention a few. SIFT, GIST and CENTRIST are well known descriptors in scene categorization. SIFT (Scale-Invariant Feature Transform) is useful for determining an object is the same object, even though it appears under different conditions (33) (35). For instance, the distinguishing key points of objects in an image can still be found when the image is scaled, as shown in this OpenCV tutorial (34), and used in cross matching between different views or scenes. However, SWIFT is not well suited for recognizing topological locations nor scene categories. The GIST descriptor on the other hand achieves high accuracy in identifying natural scene categories (36), while CENTRIST is suitable for recognizing both topological places and scene categories (33). Scene categorization as described above can be a feasible alternative to image classification using a CNN.

# Chapter 3

## Procedure and Implementation

This chapter covers the implementations of Sen. It goes over a design overview detailing Sen's intended features and includes a description of the tools utilized in implementing these features. Finally, it goes through a chronological report of the implementation of these features.

### 3.1 Design Overview

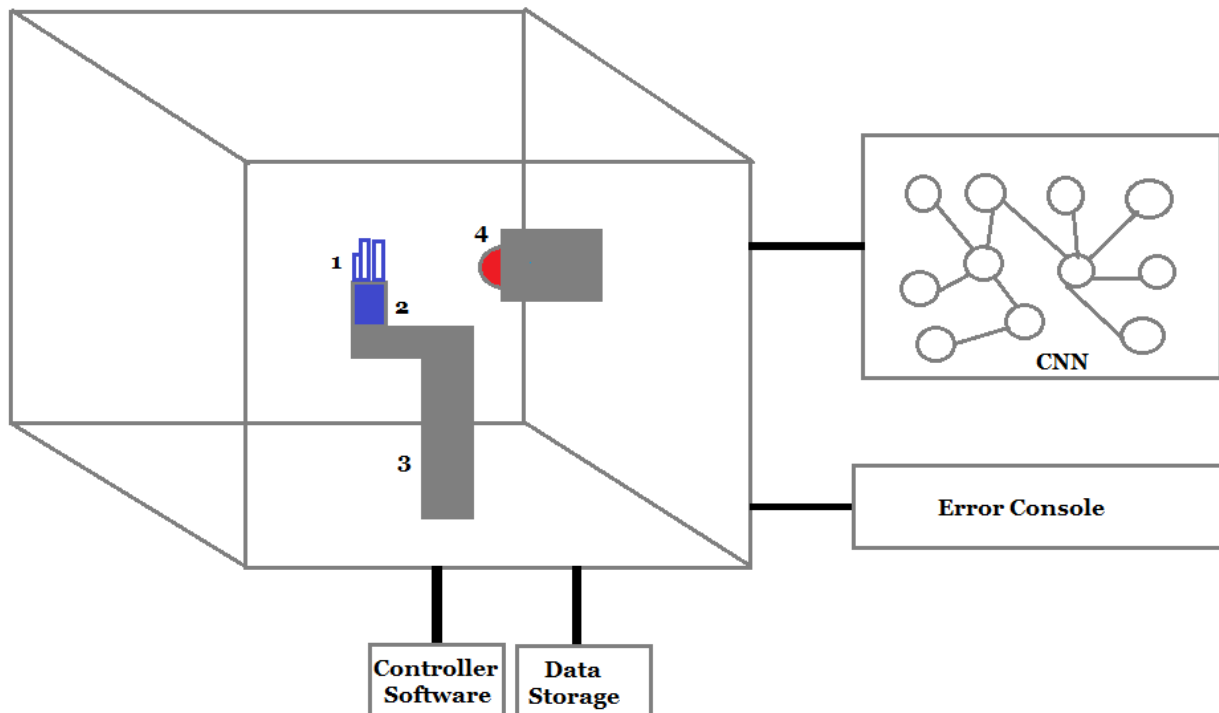


Figure 1: Design overview of Sen. 1: tactile sensors 2: anthropomorphic robotic hand 3: robotic arm 4: RGB-D camera. Black connectors represent interfaces.

Sen's features comprise:

- An interface to control a simulated robotic arm and multi-digit anthropomorphic robotic hand.
- An interface to a simulated RGB-D camera through which images taken from within the simulation environment can be saved.
- An interface to simulated tactile sensors through which contact and force measurements taken from within the simulation environment can be saved.
- A simple interface to interact with Convolutional Neural Networks.
- A simple method for viewing error messages.

## Representation of Tactile Data

Tactile data is obtained from grasp actions performed by the anthropomorphic robotic hand. The data is represented, as shown in figure 2, as a list of contact points mapped to deformation forces at each point. By deformation force it is meant the force at which a major deviation from the original point of contact occurs or the maximum force permitted by the hand to apply on an object, whichever occurs first. No specific major deviation threshold is fixed, nor maximum permitted force specified. The two parameters are variable.

Tactile Data Structure		
Location	contact point as x y z coordinate	Threshold force before deformation in Newtons
0		
1		
..		
n		

Figure 2: Representation of Tactile Data as a Data Structure

## Example Usage

A user wants to test if software that has been built for physical hardware has any undetected bugs that would damage the physical hardware. Say the piece of software automates the task of grasping and releasing an object a thousand times, while storing the tactile data of each grasp. Furthermore, say after each grasp a camera is used to determine the disturbed position of the object which is fed back to the hand in preparation for the next grasp. Using Sen this user will be able to discover bugs in their software without damaging their physical hardware. The user can test their grasp automation software by plugging into Sen's control interface. Also, the object location tracking functionality can be tested by running images from Sen's simulated camera through the actual object position tracking code. Finally, simulated tactile data from each grasp can be retrieved from Sen's data storage for any analysis the user may want to perform.

### 3.1.1 Consideration of Hardware Components

The considerations of hardware components were made within the constraints of using hardware available in the QMUL Robotics Lab. Developing Sen with this in mind allowed for developing a system that can be improved and used by future students working with the same

hardware. In summary the components which constrained and directed the development of Sen were:

1. UR5 robotic arm. See (63) for a specification.
2. Allegro robotic hand. See (64) for a specification.
3. Microsoft Kinect. See (65) for a specification.
4. Custom built tactile sensors.

Custom built tactile sensors because the Allegro hand does not come pre-fitted with tactile sensors and it was decided that custom built sensors would be developed for it. However, that significant endeavor fell outside the scope of this project's aims and while waiting for its completion the tactile data gathering aspect of this project was done entirely through simulated sensors.

### 3.1.2 Consideration of Software Libraries and Frameworks

The following factors, in no particular order, were kept in mind when considering software tools:

- Ease of replication and future development of Sen.
- Availability of getting-started open-source packages and code bases.
- Quality of documentation.
- Stability and duration of support.
- Technical requirement of the project.

One of the technical requirements of Sen is providing an interface between multiple hardware components. Two main options came up when considering a good framework for achieving this: ROS<sup>2</sup> and YARP<sup>3</sup>. ROS was ultimately chosen over YARP because of its wide community of users who provide useful documentation and error fixes, its regular updates and long-term support as well as its seamless integration with software packages used for simulation and kinematics control – something YARP did not seem to have.

With regards to simulation and control, especially path planning, ROS comes with seamless integration to various software packages designed for these tasks. Such as Gazebo (38), Rviz (39), MRPT<sup>4</sup>, OROCOS<sup>5</sup>, Matlab Robotics Toolbox, Moveit! etc. In summary, Gazebo and Rviz were chosen as the software packages for running simulations while Moveit! was chosen for path planning and control of the UR5 – something which requires kinematics calculations.

Finally, for working with Convolutional Neural Networks, TensorFlow was chosen in place of OpenCV because both were considered equally sufficient and TensorFlow was more familiar to use.

### 3.1.3 Description of Chosen Components and Software

#### 3.1.3.1 ROS

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms (37).

#### 3.1.3.2 Gazebo

---

<sup>2</sup> Robot Operating System

<sup>3</sup> Yet Another Robot Platform

<sup>4</sup> Mobile Robot Programming Toolkit

<sup>5</sup> Open Robot Control Software

Gazebo is a simulator that makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI systems using realistic scenarios (38). Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. It features include a robust physics engine, high-quality graphics, and convenient programmatic and graphical interfaces.

### 3.1.3.3 Rviz

Rviz is a 3D visualization tool for displaying sensor data and state information from Robotics Operating System ROS (39). Rviz can also display live representations of sensor values coming over ROS Topics including camera data, infrared distance measurements, sonar data and more.

### 3.1.3.4 MoveIt!

MoveIt! is a motion planning framework for mobile manipulation, incorporating 3D perception, kinematics, control and navigation (43). It provides an easy-to-use platform for developing advanced robotics applications, evaluating new robot designs and building integrated robotics products for industrial, commercial, R&D and other domains. It is used in building Cen's control interface and can be integrated with ROS and Rviz to control Cen's Gazebo simulations.

### 3.1.3.5 UR5 Robotic Arm

The lightweight, flexible, and collaborative UR5 industrial robot from Universal Robots lets you automate repetitive and dangerous tasks with payloads of up to 5 kg. The UR5 is ideal for optimizing low-weight collaborative processes, such as picking, placing, and testing (42).



### 3.1.3.6 Microsoft Kinect

The Kinect is a Color VGA video camera, RGB camera, depth sensor and multi-array microphone (44).

### 3.1.3.7 Allegro robotic hand

Allegro is a low -cost and highly adaptive robotic hand with four fingers and sixteen independent torque-controlled joints, it's the perfect platform for grasp and manipulation research (40).

### 3.1.3.8 TensorFlow

TensorFlow™ is an open source software library for high performance numerical computation (41). Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains.

## 3.2 Chronological Order of Implementation

### 3.2.1 Choosing an Operating System

At first the decision was made to use ROS Kinetic as it was the latest ROS version and had support for the newest features in other software packages such as Gazebo and Moveit!. This singular decision influenced the decision for versions of other software which were used. As ROS versions are tightly coupled to Linux versions, the project had to be built on the Ubuntu 16.04 platform, the Linux version for ROS Kinetic. Furthermore, ROS versions are also tightly coupled with gazebo versions (45), and ROS Kinetic came with Gazebo 7. As for choice of TensorFlow, the latest stable version was chosen. It required Python 3, which is itself one of the latest versions of Python. It also required a 64bit operating system. Therefore the operating system used when starting out the Ubuntu 16.04 64bit OS<sup>6</sup>.

### 3.2.2 Setting up UR5 Simulation in Gazebo

The UR5 packages come in ROS as part of an extensive universal-robot stack (42) which includes sub-packages for launching a UR5 simulation in Gazebo, planning movement paths for its end-effector, sending messages, as well as working out its kinematics. The options for installing these packages were either through the use of apt-get for Linux distributions only up to Ubuntu 14.04, or by compiling from source for higher Linux distros like Ubuntu 16.04. This is important to note. Initially an attempt was made to install the UR5 packages for Ubuntu 16.04 using the apt-get option. This had the result of seeming to install properly but went on to run with numerous errors. A lot of time was wasted in the first weeks because of this, until long sessions of debugging led to the realization that the packages for Ubuntu 16.04 had to be

---

<sup>6</sup> Operating System

compiled from source. Once this was done everything ran as expected and the project moved onto fitting the newly simulated UR5 arm with the Allegro hand.

### 3.2.3 Fitting the UR5 with the Allegro Hand

The existing ROS Allegro hand stack was built for ROS versions up to Indigo. There were two options to choose from, one developed by the creators of Allegro; Simlabrobotics (47), or another which had been developed by the open-source community past what Simlabrobotics provided (46). The latter was chosen. This was chosen chiefly because it was more developed than Simlabrobotics' branch. However, on testing it on ROS Kinetic, although a good visualization was available in Rviz, the Gazebo model was completely broken, most likely because of an incompatible SDF file which stemmed from the fact that it was generated from a URDF written for Indigo and URDFs are not designed to be backward compatible (48). This problem was encountered early on in the project and it was thought that fixing this would be possible over a three-week period, especially with help from the original developer, so an attempt was made to do so.

### 3.2.5 Challenges Faced in Fixing the Allegro Simulation

The first attempts made at fixing the Allegro Gazebo simulation simply involved researching the error messages available from the Rviz Graphical User Interface. The main error messages were along the line of:

```
``` no tf transform available between link [some link] and [some_other_link]```
```

This had to do with how the links were attached by joints in the URDF and subsequently in the SDF generated from the URDF. A better understanding of the nature of this problem can be found here (49). There were two main options to solving this: modifying the existing URDFs to conform to the Kinetic version or building an entirely new model for Kinetic.

The former proved challenging to implement because at the time there was a limited understanding of how to work with URDFs. An attempt was made to find a URDF updater which

converts old URDF files to new URDF files, but this was not readily available. Further attempts were made to contact the original developer, Felix Duvallet, to see if he would be willing to update this aspect of his work. Although he was successfully contacted, he was unable to spare the time required to do this, giving the impression that even for him it would involve considerable time.

As a result, the first discussion on rescoping this project was held with the project supervisor. The option was to take on the task of creating a new Allegro model for Kinetic and Gazebo 7 as a project, or perhaps even build a URDF upgrade tool, but ultimately the decision was made to continue with the aims of the current project by using another end effector other than the Allegro Hand with a view to going back to the existing problem at a later date.

To prevent similar problems and based on the observation that Kinetic was so new that open-source developers had not yet caught up to it, the decision was made to carry out future development on Indigo and the Ubuntu 14.04 64 bit platform. With the aim of having a working system built as quickly as possible that could then be upgraded to work on newer platforms. As a result, a convenient way of switching between operating systems without affecting the natively installed system was needed.

### 3.2.6 Choosing to Work in VM for Speed

Virtual Machine technology was chosen for switching to the ROS Indigo in place of dual booting or other alternatives such as containerization technology. The VM software used was Virtual Box version 5.2.8. Note that on a Windows 10 OS with 7GB Ram most of the development work done in the VM was able to run smoothly until work began on using MoveIt!. Among other things, a VM allowed for quick restarts whenever the environment got complicated, such as when different versions of python had been installed and broken but running builds of ROS packages were causing endless streams of error messages. A VM image with a simple Ubuntu and ROS install cut down the time it took to restart in such situations, which in turn sped up development of the first subscribers and publishers that were eventually to become the control interface of the UR5 arm. At this point the following had been achieved:

- The delivery of a shareable virtual environment containing all that was needed to quickly get started developing for ROS. Which is something that would have been helpful when the project began.
- A system which fed camera input from an external camera onto a ROS topic<sup>7</sup> that was subscribed to by an image classifier which determined what numbers were displayed on the images being published to the ROS topic. This interface is useful for real-world testing of camera software and is the second reason why it was worked on, along with the usefulness of creating a way to interface ROS with an image classifier.

### 3.2.7 Real World Camera System

The aim was to get pictures from a webcam which were published onto a ROS topic<sup>7</sup> and classified using a CNN which retrieved the pictures from the ROS topic. The work was based off an already existing implementation found on Github<sup>8</sup> (52):

There was a problem with getting camera feeds from a webcam while using the Ubuntu 14.04 64 bit VM on a windows host. To solve this Droidcam was used, which provided the solution of streaming camera images from a phone via a USB or WiFi connection to a USB port which was accessible in the VM.

Using `cv_camera` as used in (52), the input stream of images was published on a ROS topic from which they were accessed by an number image classifier node. The results were accessed from another ROS topic, `/result`, specified in line 81 of `tensorflow_in_ros_mnist.py` from (52).

This confirmed that an interface to a real-world camera could be easily established and used to provide input to a convolutional neural network. However, a simulated camera looking into the gazebo simulation environment still had to be created for Sen.

---

<sup>7</sup> ROS Topics are named busses over which ROS nodes exchange messages (50)

<sup>8</sup> GitHub is a web-based hosting service for version control using git (51).



### 3.2.9 Creating a UR5 Controller

The main advantages of choosing ROS is that it has seamless integration with Moveit! which handles solving quite tedious kinematics equations. (53) explains how to start motion planning with Moveit!

The aim of the interface was to create an alternative to the Rviz GUI for controlling the UR5 simulation. A simple ROS package was developed in the ROS indigo virtual environment following (53) which takes a cartesian position as its argument and moves the UR5 end-effector to that position.

Some issues arose such as the need to understand how asynchronous mechanisms allow for non-blocking code, and the fact that some of the documentation was outdated, specifically the information to do with the commands needed to actually move the end effector into place after the required joint parameters had been solved and a path to moving them into the desired positions found. In summary, asynchronous spinners are needed to prevent certain `move_group` functions from blocking and the commands used to actually move the simulation can be seen in the code for this project.

Other than these, once a good grasp of the ROS subscribers and publishers was established by going through a course provided by ETH Zurich (54), it was possible to build an interface that not only moved the end effector to a desired position but could also move it along a path. At this point the desired end effector position was sent via a terminal by running the **rostopic pub** command to publish cartesian coordinates to the relevant topic.

### 3.2.10 Building a Position Publisher Node

Having successfully built a ROS package to move the UR5 into a desired location, a publisher was coded which publishes end effector coordinate locations to the topic on which the positioner node listened. The reasoning was that this package would be used to

form a closed loop if it contained code that relied on the output from a CNN to determine how the end effector should be positioned. Though built and available for further development, this package was dropped from further use.

### 3.2.11 Including a Camera in Gazebo

With preliminary infrastructure for the UR5 control interface in place, the next task was to create a camera feed which published images taken from inside the gazebo environment onto a topic from which the images could be read.

Two approaches were made in doing this. In the first approach a camera plugin was built – based off (56), because it had been learned that a plugin offered the advantage of quick image retrieval (55). This is because it would be possible to retrieve the images from memory without the need to store them as files when using a plugin. Which could be potentially useful in future applications of Sen, say in the case when a user requires very high-speed tests. A plugin also offers easy portability across projects, in that as an independent ROS plugin can easily be added to any compatible ROS project and used in taking pictures.

However, plugins were ROS version dependent, and at this point development was still being carried out in ROS Indigo and Gazebo 2 on the VM, so the camera plugin development work had to be postponed until the development environment was upgraded from ROS Indigo to the intended environment of ROS kinetic.

In the second approach a camera sensor was added onto a robot and placed in gazebo environment. This offered the disadvantage of being slower in image retrieval but the advantage of being faster to implement than the camera plugin as ample material on using this approach was easily available (57) and so it was chosen as the interim method for including a camera, pending future development of a plugin.

Some challenges were faced. At this point development work was still being carried out in a virtual environment. Unfortunately, it turned out that there was a known problem with streaming camera images from a camera sensor in gazebo while working in virtual



environments which was solved by working on a native install. After discovering this it was time to upgrade all the work done so far to ROS Kinetic, so that development could continue on native installs available in the robotics lab.

### 3.2.12 Upgrading Packages to Kinetic

The changes in the Moveit! movegroup libraries from Indigo to Kinetic were substantial. Most of the function names had been changed, and so the C++ implementation of a UR5 control interface that had been written had to be re-written. Furthermore, the CMakeLists and package.xml files had to be updated to work with Kinetic. The UR5 packages in use were also updated to the Kinetic versions. The camera sensor packages were also updated to use Kinetic.

### 3.2.13 Successfully Streaming Simulated Camera Images

Once the packages in use had been updated it was possible to run the UR5 simulation, the interface and the camera sensor on a native install of Ubuntu 16.04 which solved the issue with streaming camera images from a camera sensor. This paved the way for integrating an image recognition node with the existing system. It was a good opportunity to check that images taken in the gazebo environment were of sufficient quality to be used as test data or even training data for CNNs such as the ones that were integrated with Sen.

It was possible to easily integrate two CNNs with Sen by having them subscribe to ROS topics or writing special bash scripts to provide them with the images that had been taken from the Gazebo environment. The first CNN was a number classification node while the second was an image recognition node. The results from the second CNN are included in the results section.

### 3.2.14 Publishing End-Effector Positions with Python

The final task accomplished within the time allocated for this project was to modify the position publisher package to involve the use of keyboard inputs, much like the teleop-twist input interface for the turtle-bot robot.

U J for up down along the vertical axis

W S for up down along the Y axis

A D for left right along the X axis

X to quit

The interface does not include input keys for orientation, but these can be published using rostopic pub directly to the relevant topic.

### 3.2.15 Research into Gathering Force and Contact Data from Gazebo

Cursory research had been made before building the test environment into how contact and force data from grasp actions would be gotten from the gazebo environment. The findings of the research are included in the results section.

# Chapter 5

## Results and Contribution

### 5.1 Image Classification Results

For Figures 3 - 6 the top five results of classification are presented after each figure. The classification was performed using TensorFlow's Inception-v3 classifier (60). The score associated with each class can be interpreted as level of certainty that the image belongs to that class.

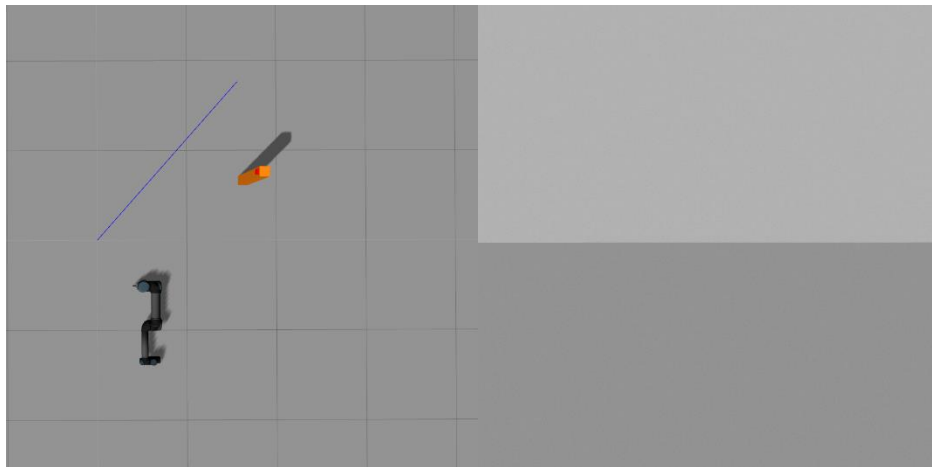


Figure 3: On the left: top view from Gazebo. On the right: view from camera looking onto horizon

envelope (score = 0.10778)

letter opener, paper knife, paperknife (score = 0.02310)

stopwatch, stop watch (score = 0.02072)

rule, ruler (score = 0.02055)

nematode, nematode worm, roundworm (score = 0.01702)

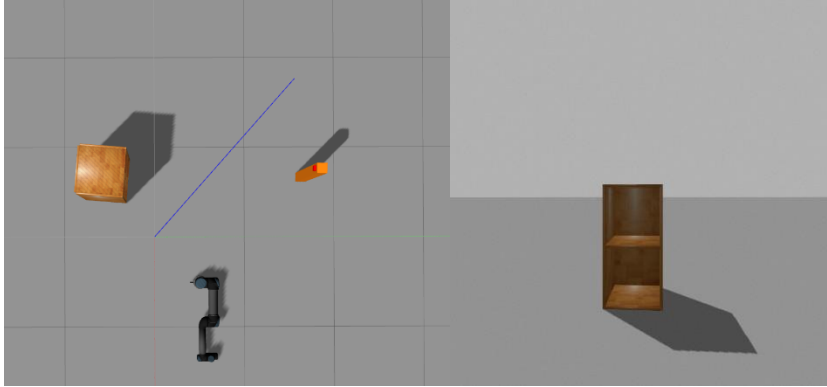


Figure 4: On the left: top view from Gazebo. On the right: view from camera looking onto a shelf

bookcase (score = 0.20354)

wardrobe, closet, press (score = 0.12303)

medicine chest, medicine cabinet (score = 0.08154)

safe (score = 0.02761)

file, file cabinet, filing cabinet (score = 0.02519)

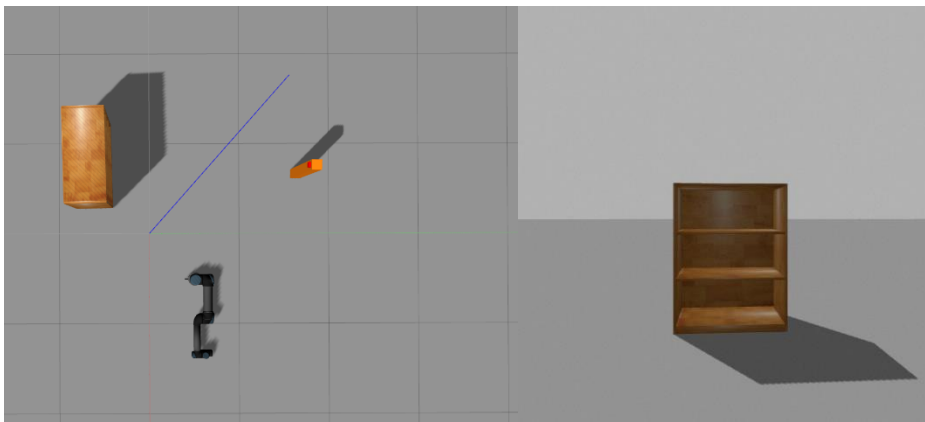


Figure 5: On the left: top view from Gazebo. On the right: view from camera looking onto a wide shelf

bookcase (score = 0.26374)

wardrobe, closet, press (score = 0.13302)

medicine chest, medicine cabinet (score = 0.06216)

chiffonier, commode (score = 0.03650)

file, file cabinet, filing cabinet (score = 0.03352)

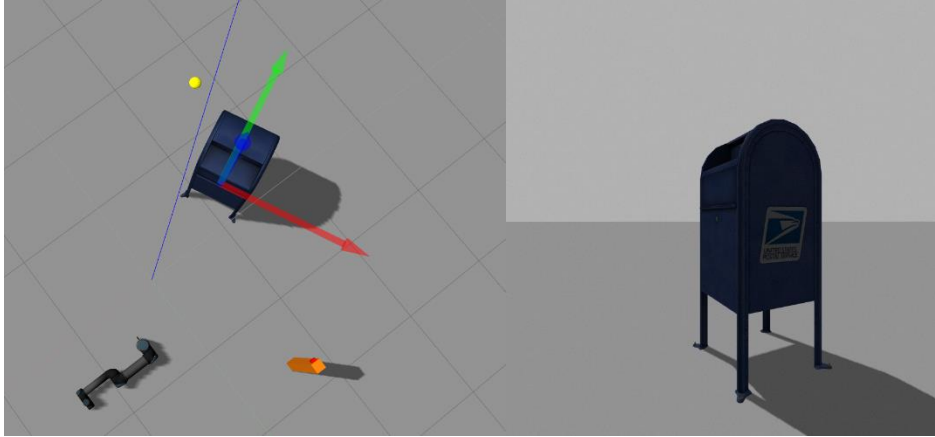


Figure 6: On the left: top view from Gazebo. On the right view from camera looking onto a mailbox

mailbox, letter box (score = 0.99597)

backpack, back pack, knapsack, packsack, rucksack, haversack (score = 0.00010)

bassinet (score = 0.00010)

cradle (score = 0.00005)

chest (score = 0.00004)

## 5.2 Stored Image Files and CNN Output Log File

Figure 7 shows a screenshot of images as well as the log file containing their classification output which have been automatically saved while using Sen. In this instance 301 images were taken using Sen's camera and a classification log file of 64KB was generated. It took approximately 5 minutes to generate this data.

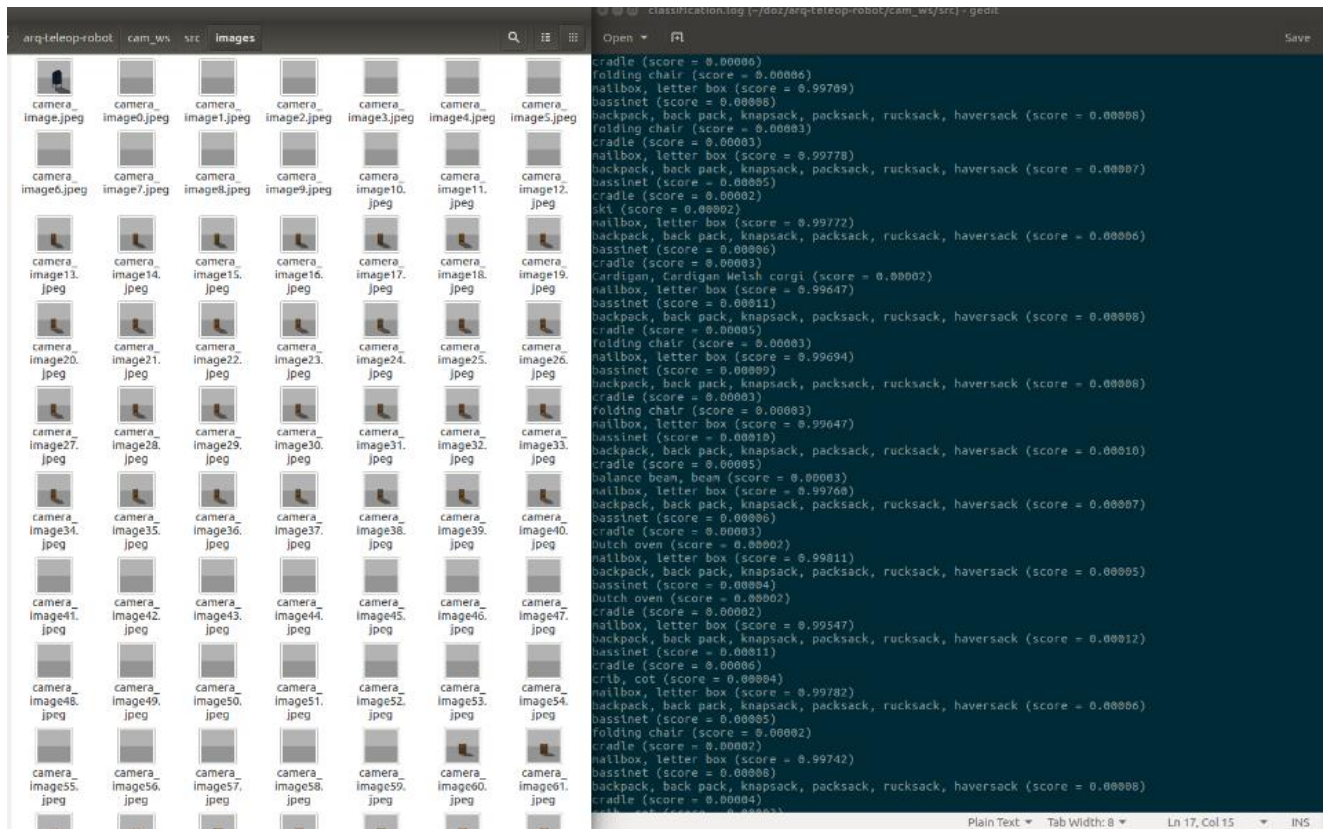


Figure 7: Images and log file that have been stored automatically

## 5.3 Testing the Python Control Interface

Here images from a test of the keyboard interface to see if it can be used to position the UR5 end-effector above a cinder block are provided. In the top left of each image a terminal shows a list of statements where each statement corresponds to a single keyboard command. In Figure 8 the terminal output shows that it has taken 14 movement commands to position the UR5 arm where it is shown positioned.

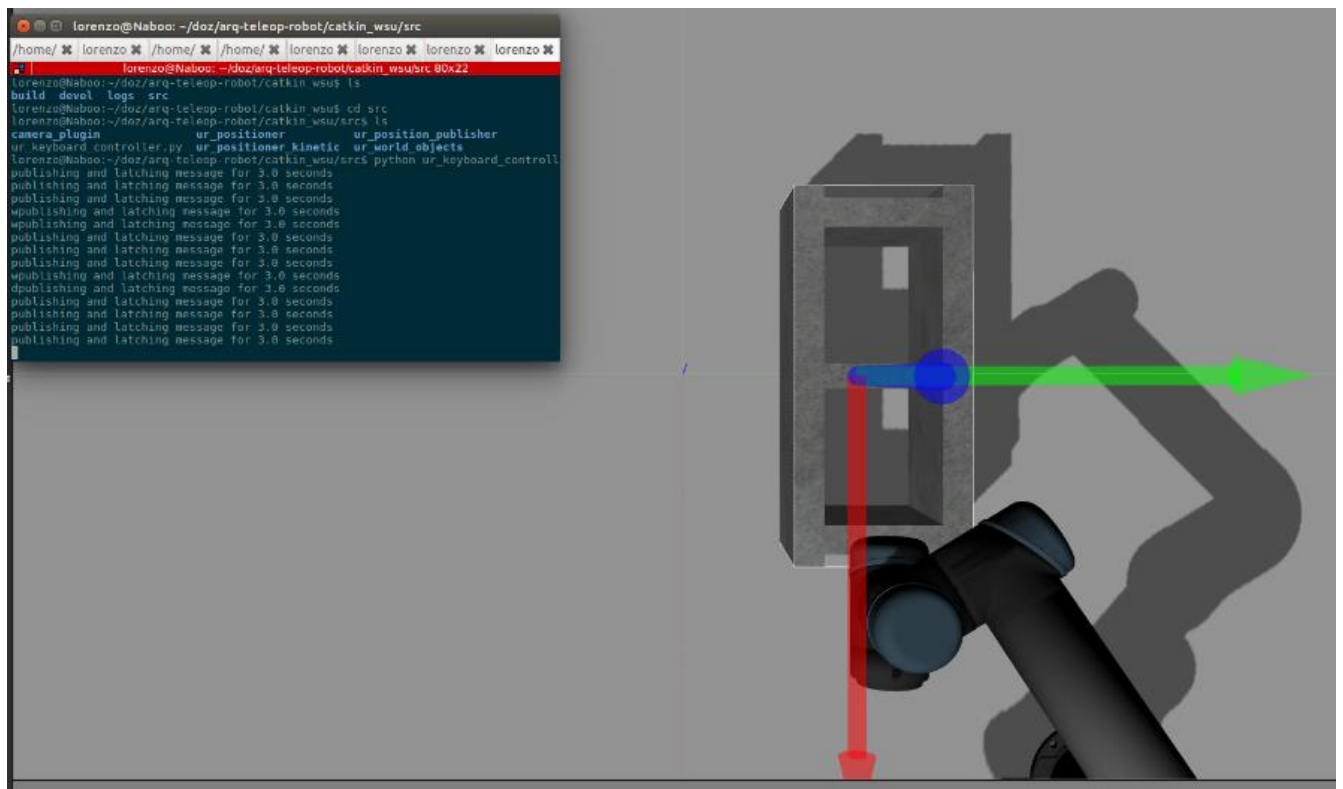


Figure 8: UR5 arm being moved into position above a cinder block

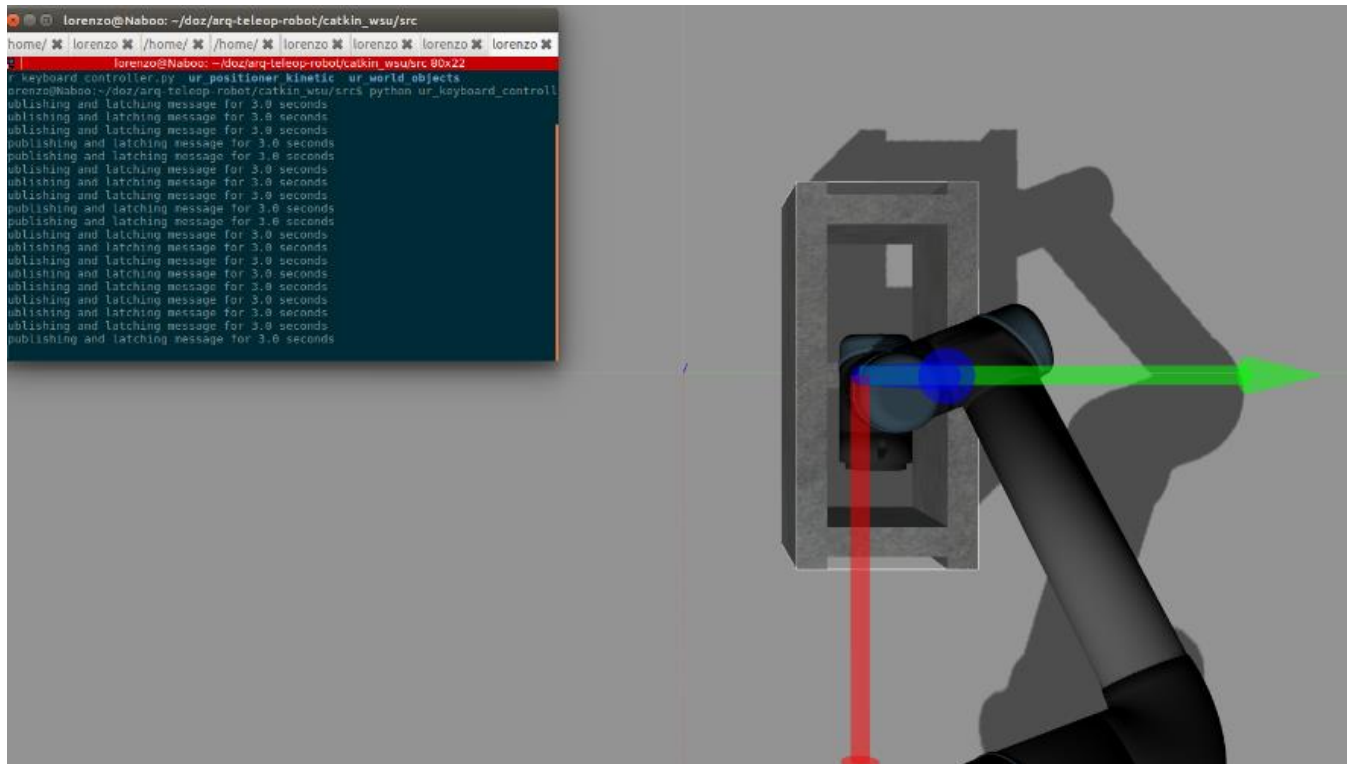


Figure 9: UR5 in desired position above cinder block



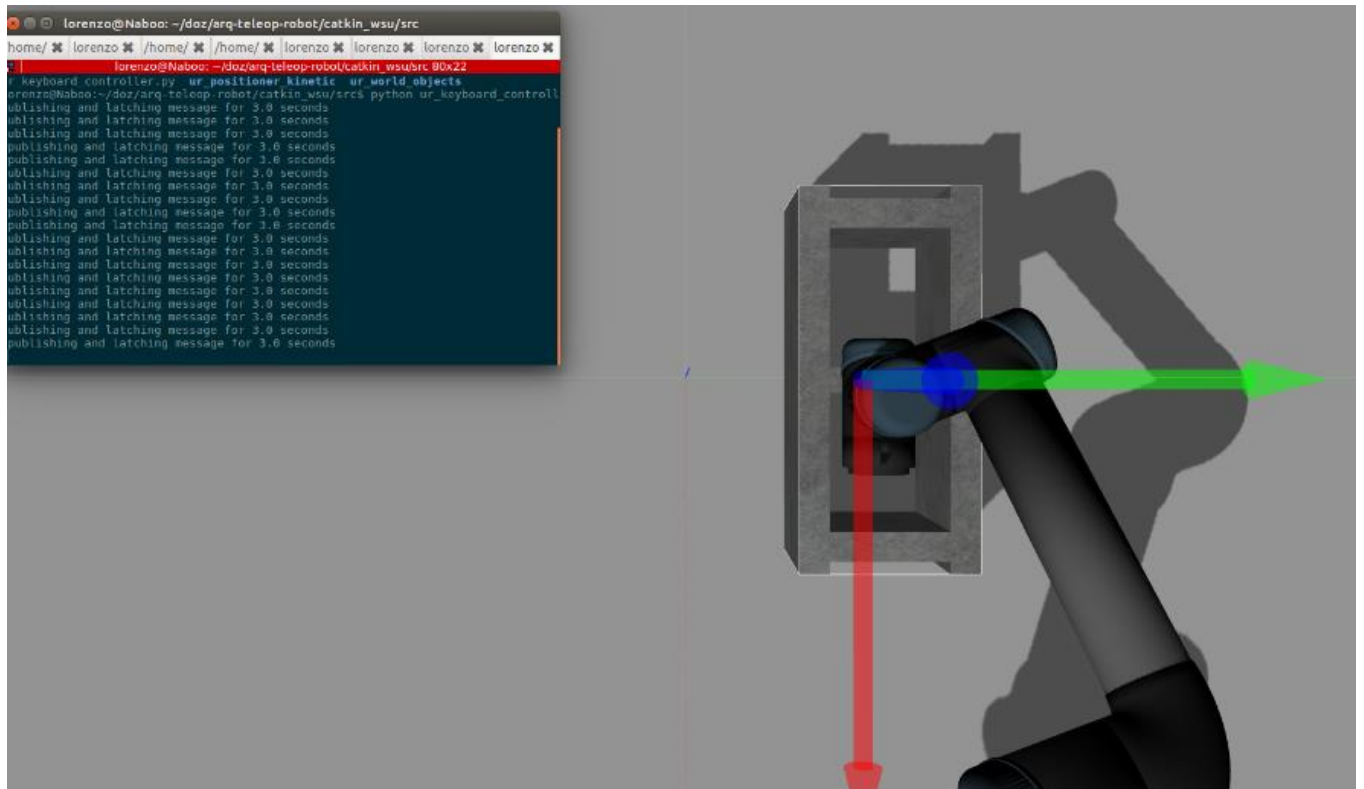


Figure 10: UR5 moved downwards to collide with cinder block

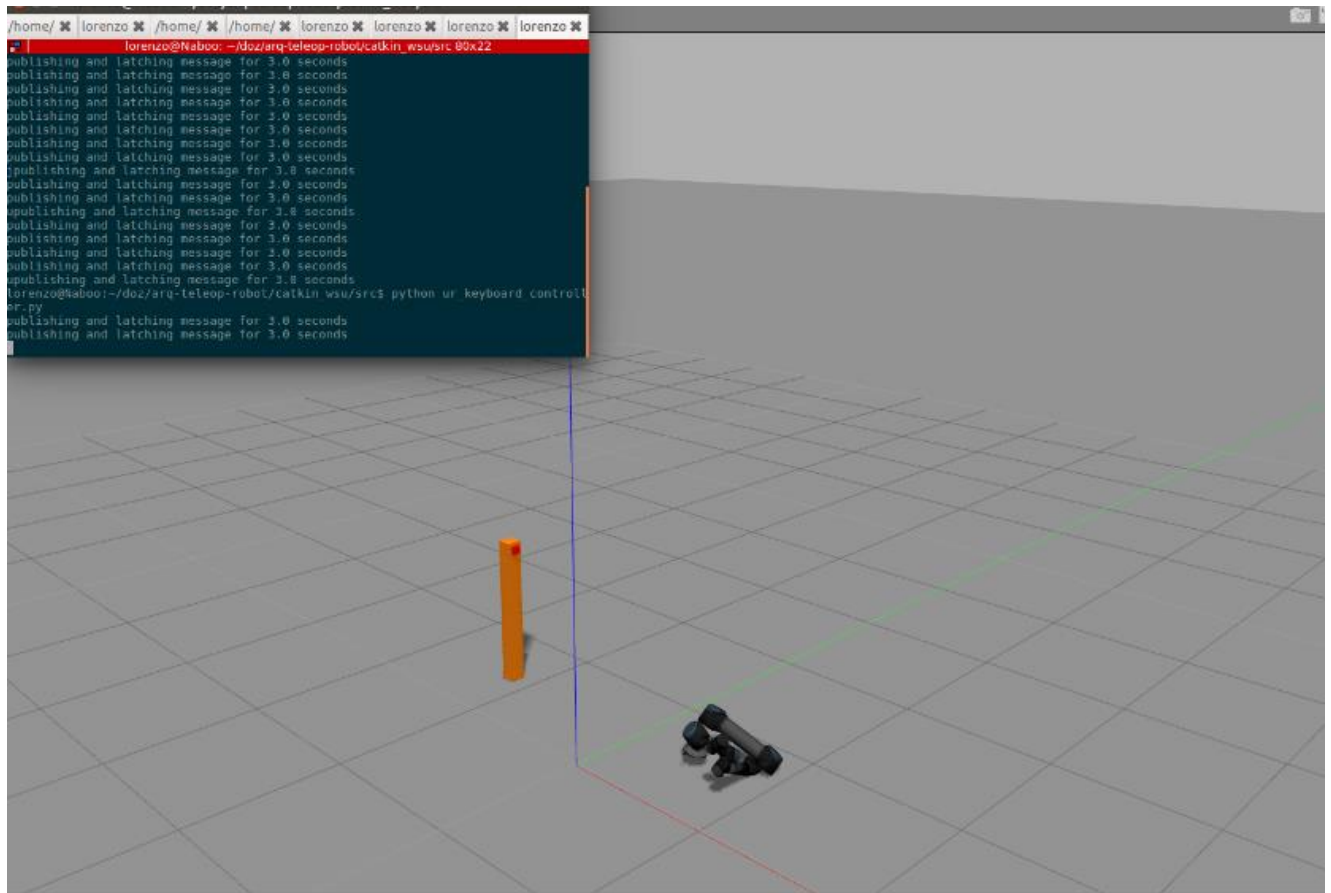


Figure 11: Damaged UR5, similar to what happens when it collides with a cinder block

The effect shown in Figure 11 occurs sometimes when the UR5 moves into its control position. The control interface is set to always move the UR5 into the same reference position when the interface is first started. Sometimes the path planned to move the UR5 into this position takes it into collision with the ground.

## 5.4 Error Messages

Included below are prominent error messages of errors which occurred when testing Sen. For most errors a simple restart of Sen sufficed.

### 5.4.1 Error When Sen Runs as Expected

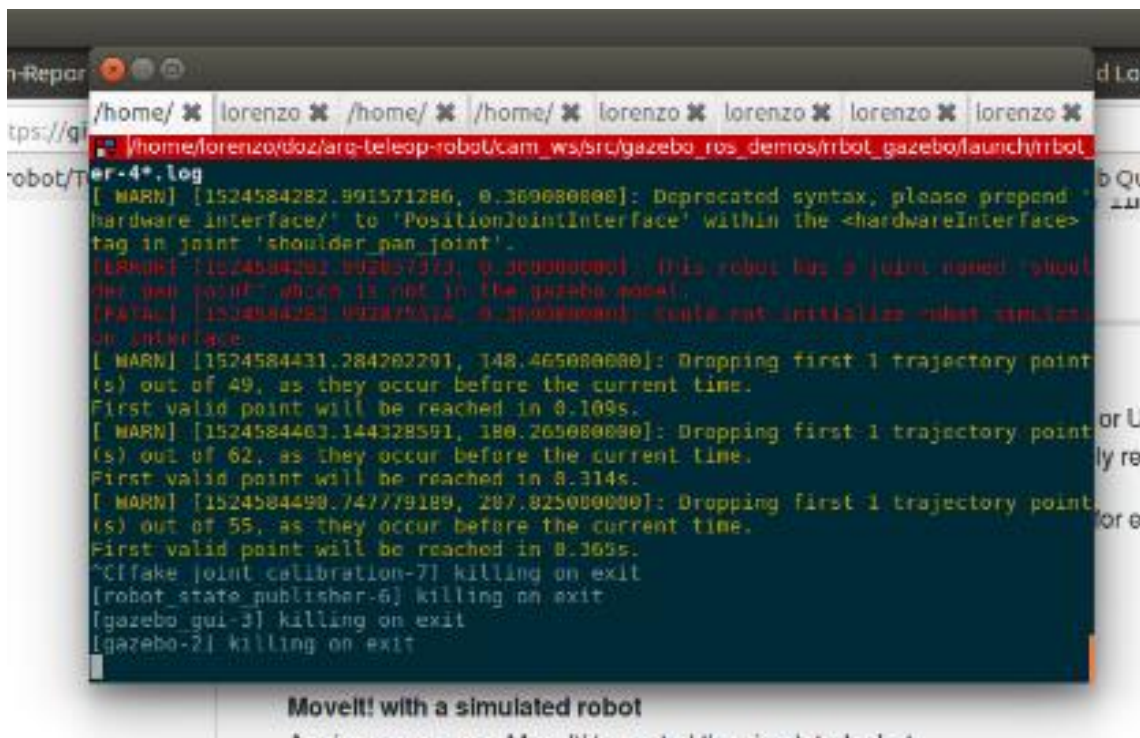
A terminal window with a dark background and light text. The window title bar shows several tabs, with the active one displaying the path: `/home/lorenzo/catkin_ws/src/gazebo_ros_demos/robot_gazebo/launch/robot_gazebo.launch`. The terminal output shows ROS launch logs. Several lines are highlighted in red, indicating error messages. The errors include a warning about deprecated syntax for a joint interface, a fatal error about a joint named 'shoulder\_pan joint' not being in the Gazebo model, and a fatal error about not being able to initialize the robot simulation interface. There are also several warning messages about dropping trajectory points. At the bottom of the terminal output, there are several 'killing on exit' messages for various processes like 'fake\_joint\_calibration-7', 'robot\_state\_publisher-6', 'gazebo\_gui-3', and 'gazebo-2'. Below the terminal window, there is a small white box with the text 'MoveIt! with a simulated robot'.

Figure 12: Error message on launching Sen's world file

Figure 12 shows error messages (in red) that occur even when Sen is running as expected:

- This robot has a joint named `shoulder_pan joint` which is not in the gazebo model.
- Could not initialize robot simulation interface

### 5.4.2 Segmentation Fault

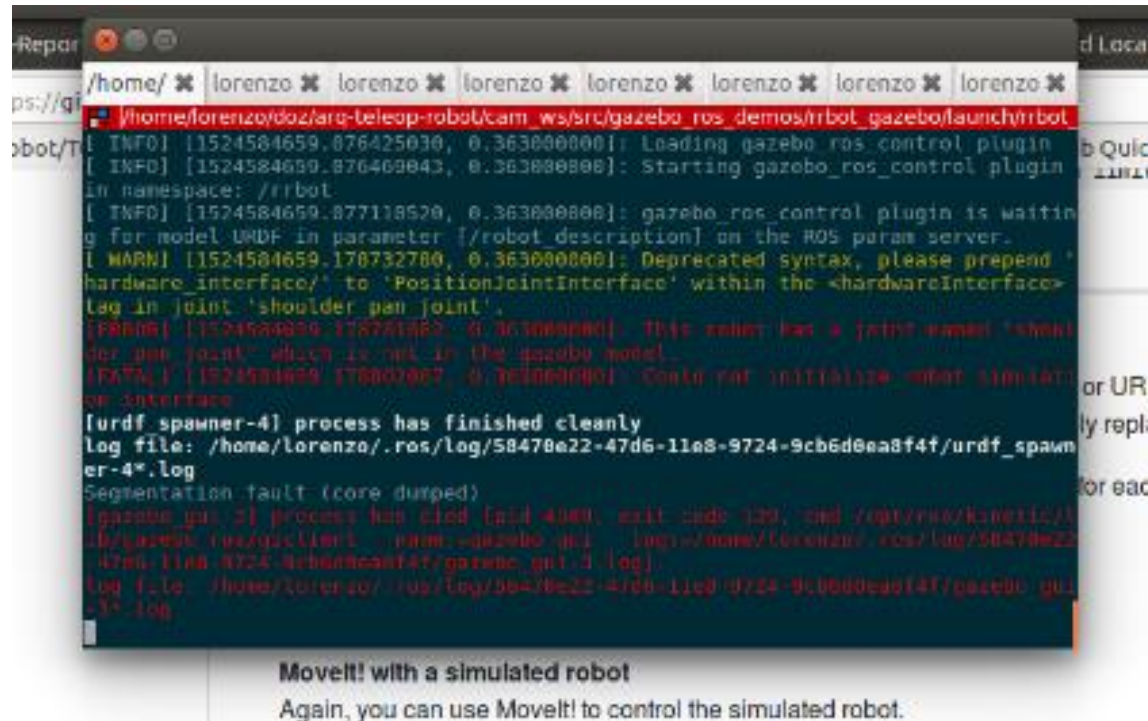


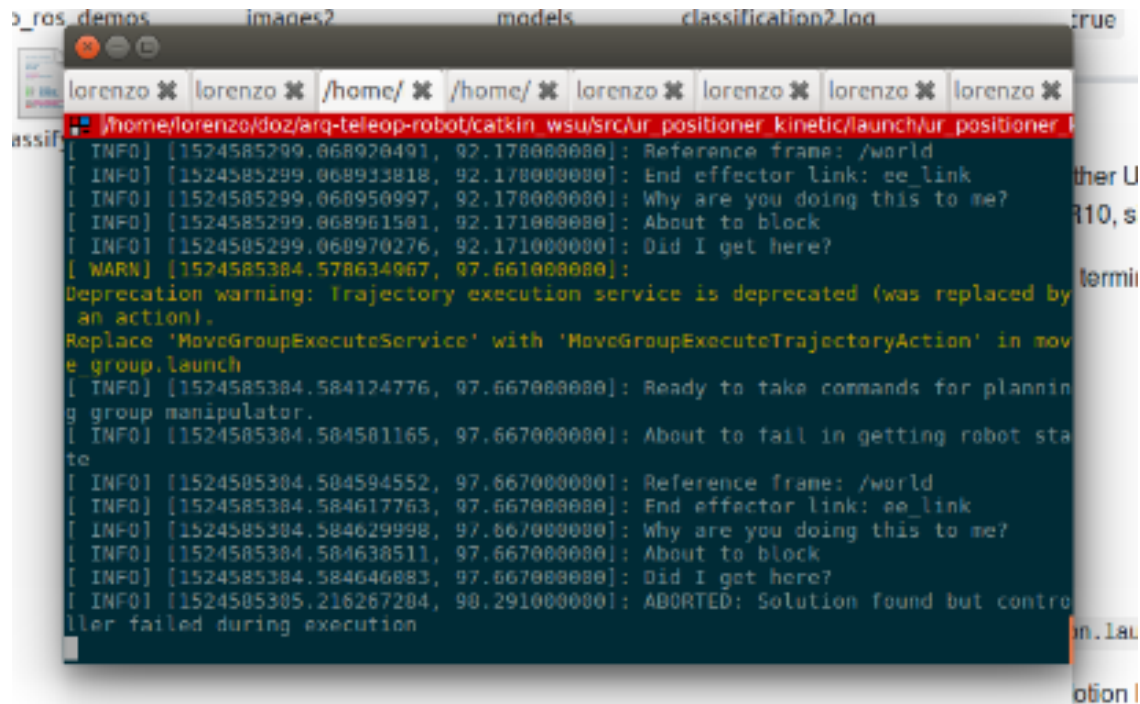
Figure 13: Error message from the Gazebo process when a segmentation fault occurs

Figure 13 shows an error message that occurred indicating there had been a segmentation fault:

- Segmentation fault (core dumped)

This error proved difficult to reproduce.

### 5.4.3 Errors When Collisions Occur



```
lorenzo ✖ lorenzo ✖ /home/ ✖ /home/ ✖ lorenzo ✖ lorenzo ✖ lorenzo ✖ lorenzo ✖
[ INFO ] [1524585299.068920491, 92.178000000]: Reference frame: /world
[ INFO ] [1524585299.068913818, 92.178000000]: End effector link: ee_link
[ INFO ] [1524585299.068950997, 92.178000000]: Why are you doing this to me?
[ INFO ] [1524585299.068961501, 92.171000000]: About to block
[ INFO ] [1524585299.068970276, 92.171000000]: Did I get here?
[ WARN ] [1524585384.578634967, 97.661000000]:
Deprecation warning: Trajectory execution service is deprecated (was replaced by
an action).
Replace 'MoveGroupExecuteService' with 'MoveGroupExecuteTrajectoryAction' in mov
e_group.launch
[ INFO ] [1524585384.584124776, 97.667000000]: Ready to take commands for plannin
g group manipulator.
[ INFO ] [1524585384.584501165, 97.667000000]: About to fail in getting robot sta
te
[ INFO ] [1524585384.584594552, 97.667000000]: Reference frame: /world
[ INFO ] [1524585384.584617763, 97.667000000]: End effector link: ee_link
[ INFO ] [1524585384.584629998, 97.667000000]: Why are you doing this to me?
[ INFO ] [1524585384.584638511, 97.667000000]: About to block
[ INFO ] [1524585384.584646083, 97.667000000]: Did I get here?
[ INFO ] [1524585385.216267204, 98.291000000]: ABORTED: Solution found but contro
ller failed during execution
```

Figure 14: ABORTED error message from the ur\_positioner\_kinetic package when a collision occurs

The last line in Figure 13 shows the message that occurs when the UR5 has collided with an object:

- ABORTED: Solution found but controller failed during execution



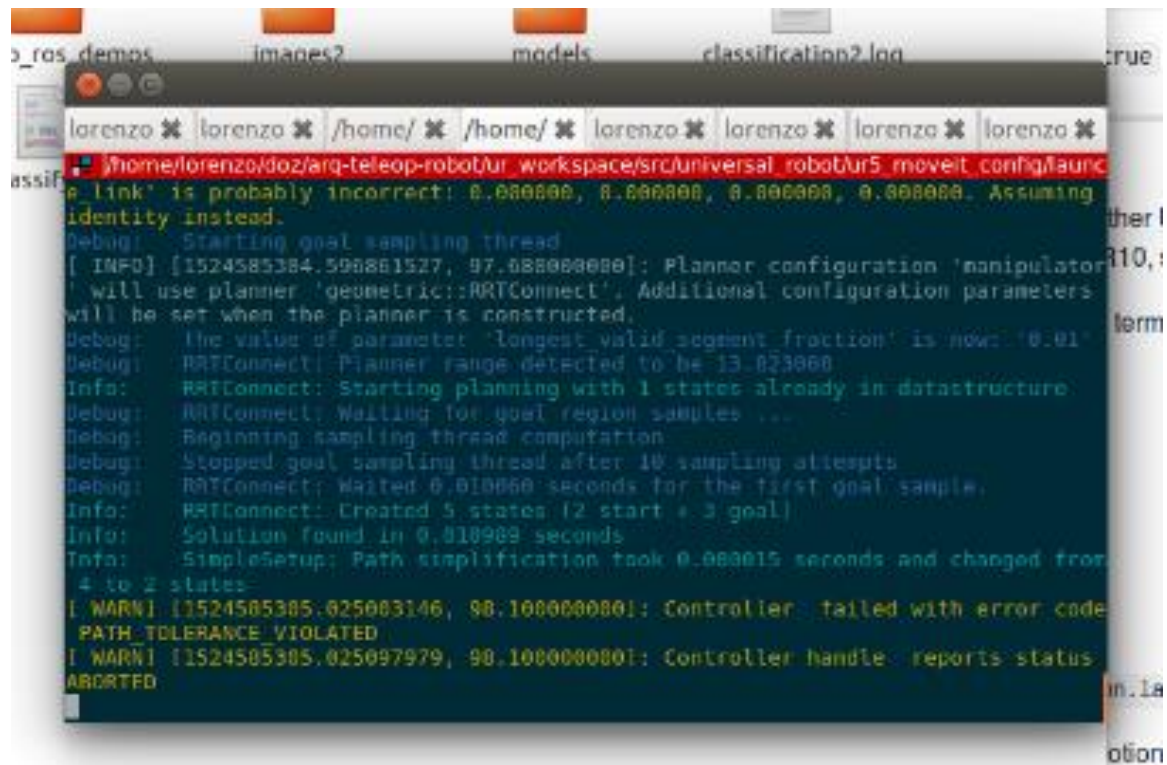
A terminal window with multiple tabs at the top: 'ros demos', 'images2', 'models', 'classification2.log', and 'true'. The active tab is 'classification2.log'. The terminal shows a series of log messages from a ROS node. The first line is a red error message: 'Error: /home/lorenzo/doz/arg-teleop-robot/ur\_workspace/src/universal\_robot/ur5\_moveit\_config/launch/ur5\_moveit\_config.launch: 'e\_link' is probably incorrect: 0.000000, 0.000000, 0.000000, 0.000000. Assuming identity instead.' This is followed by several debug and info messages from the 'RRTConnect' planner, including 'Starting goal sampling thread', 'Planner range detected to be 13.023060', 'Starting planning with 1 states already in datastructure', 'Waiting for goal region samples ...', 'Beginning sampling thread computation', 'Stopped goal sampling thread after 10 sampling attempts', 'Waited 0.010000 seconds for the first goal sample', 'Created 5 states (2 start + 3 goal)', 'Solution found in 0.010000 seconds', and 'Path simplification took 0.000015 seconds and changed from 4 to 2 states'. The final two lines are red warning messages: '[WARN] (1524585385.025003146, 90.1000000000): Controller failed with error code PATH\_TOLERANCE\_VIOLATED' and '[WARN] (1524585385.025097979, 90.1000000000): Controller handle reports status: ABORTED'.

Figure 15: Error message from the path planner when a collision occurs

Figure 15 shows errors from the path planner when a collision occurs:

- Controller failed with error code PATH\_TOLERANCE\_VIOLATED
- Controller handle reports status ABORTED

## 5.5 Memory Usage

It was of interest to note how much resources in the way of memory was used when running Sen with nearly all features in use. The readings in Figures 16 -17 were taken when Sen was storing images and writing their classification output to a log file.

```

lorenzo@Naboo: ~/doz/arq-teleop-robot/cam_ws/src 80x22
lorenzo@Naboo:~/doz/arq-teleop-robot/cam_ws/src$ rqt_image_view
^C
lorenzo@Naboo:~/doz/arq-teleop-robot/cam_ws/src$ free -m
              total        used        free      shared  buff/cache   available
Mem:          15754         3562         9141          124         3050        11660
Swap:         16211           0         16211
lorenzo@Naboo:~/doz/arq-teleop-robot/cam_ws/src$

```

Figure 16: memory usage as seen when the **free -m** command was used

```

top - 17:10:39 up 3:05, 10 users, load average: 2.41, 4.29, 3.76
Tasks: 277 total, 1 running, 276 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1.0 us, 0.2 sy, 0.0 ni, 98.1 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 16132108 total, 10536300 free, 2473656 used, 3122144 buff/cache
MiB Swap: 16601084 total, 16601084 free, 0 used, 13141296 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM    TIME+  COMMAND
 7351 lorenzo   20   0 2565992 543300 160232 S   15.3   3.4   0:47.37 Web Content
 1026 root       20   0 611828  19584  14636 S    1.3   0.1   0:04.80 NetworkManager
 6982 lorenzo   20   0 2820640 604864 131720 S    1.3   3.7   4:22.12 Web Content

```

Figure 17: memory usage as seen when the **top** command was used

## 5.6 Findings of Tactile Sensor Research

(58) (59) show that it is possible to implement tactile sensors to measure tactile data of the nature described in 3.1, using force and contact sensors in Gazebo which are positioned at joints and contact surfaces respectively.

## 5.7 Contribution

This project contributes Sen with usage instructions provided in Appendix A. Sen provides:

- a control interface to a simulated UR5 arm via a ROS package called `ur_positioner_kinetic` which solves motion-planning requests, and a Python keyboard

user interface, `ur_keyboard_controller.py`, that wraps around this ROS package. The `ur_positioner_kinetic` package and keyboard UI are located in `catkin_wsu/src`.

- an interface to access and save images via a simulated robot equipped with a camera sensor and a ROS node which subscribes to the ROS topic on which the camera robot publishes images to, then saves the images to an image folder. The robot with a camera sensor is defined in a URDF file called `rrbot.xacro` which is located in `cam_ws/src/gazebo_ros_demos/rrbot_description/urdf`. The ROS node is named `classify_gazebo_image.py` and is located in `cam_ws/src`.
- an interface to plugin Python implementations of CNNs into ROS to perform classification on images taken from the Gazebo simulation environment. The CNNs can either be configured to read images from a ROS topic using the `rospy` library, or a simple script such can be written to provide the images by wrapping around the CNNs command line interface as is done in `classify.sh`. `classify.sh` was written for Sen and is located in `cam_ws/src`. Apart from feeding in images it is also used for logging the classification output.

This project also contributes in a risk assessment guide for carrying out projects of a similar nature. This is found in Appendix B.



# Chapter 6

## Discussion

In this section the meaning of the results from Chapter 5 are discussed with a view to determining if the original aim and objectives of this project were achieved. Also, the relevance of the results to engineering problems is considered by looking at what the results say about potential for real world application of Sen. Finally, the errors encountered when running Sen are analyzed for possible solutions.

Recall from Chapter 1 that this project sought to answer the question:

In learning affordances, does tactile data combined with visual data improve the performance of a robotic system in detecting tool dependent affordances?

In answering this question a key requirement was the collection of tactile and visual data for training and testing CNNs designed for use in detecting affordances. In turn, a simulation environment to test any software developed for use in gathering the necessary data was crucial to the entire project.

### 6.1 Was the aim Achieved?

This project never advanced to the stage of measuring affordance detection when done using tactile data, visual data, or a combination of the two. Therefore, due to a lack of data it is still unconcluded whether tactile data will be useful or not in improving the detection of affordances when combined with visual data.

## 6.2 Were Basic Objectives Achieved?

The most crucial and fundamental objectives of this project, which would pave the way for rapid achievement of the objectives following them were mostly achieved. These objectives were:

- to devise a method for acquiring the necessary data using a robotic arm, a multi-digit anthropomorphic robotic hand, an RGB-D camera, and tactile sensors.
- to setup a simulation environment to test software developed for use on the tools listed above in acquiring the necessary data.

In sections 5.1 - 5.2 a method for gathering visual data using a camera is demonstrated in simulation. This method can be replicated in a real-world system. While in section 3.1 under representation of tactile data, a method for gathering tactile data is described for gathering data in a real-world system. Furthermore, the results of research into the plausibility of gathering tactile data from a Gazebo environment confirms that the simulated sensors needed to do so are available.

As for the objective of setting up a simulation environment to test data gathering software, 5.1 - 5.3 demonstrate a working version of this. This environment as earlier noted is called Sen. In 5.1 - 5.3 Sen is successfully used to test an image gathering software that saves images to a folder. While in section 5.3 Sen is used to test software for gathering tactile data even without the presence of tactile sensors.

The image gathering software test in section 5.2 shows that images gathered using a real-world implementation of Sen would be able to store up to 300 images in 5 minutes or 1 image/s. This is quite a slow rate of image storage but as discussed in section 3.2.11 accessing images from memory can be used to speed up the rate at which images are accessed, which would be useful in a real time system. However, if the images are not needed in real time this rate of image capture should be sufficient, and having a camera interfaced with ROS can make the process of mapping tactile data to visual data very straightforward.

The tactile data gathering software test which doubled as the keyboard interface test showed that Sen can be used to see when such software would be damaging to physical hardware. In figure 11 the UR5 is shown with damaged joints and in figures 14 –15 we see error messages showing that the UR5 can no longer be moved. These occurred when the UR5 was moved into collision with a surface such as a cinder block or the ground. The figures demonstrate that software which does not prevent harmful collisions in taking tactile data readings will damage the UR5 arm. But it is difficult to know when software will be safe and this is where Sen comes in handy.

## 6.3 Potential for Real World Application

This section considers the question can Sen's data be used to train and test CNNs in place of real world data? If this is possible then all the data needed for this project can be gathered through Sen. To answer the question at hand, the ability of a CNN to correctly classify images taken from simulation as demonstrated by data in section 5.1, suggests that at the very least Sen's image data can be used to test a CNN. If it can be used to test it with successful results, then this also suggests it can be used to train it for usage on real-world image data.

## 6.4 Errors Encountered when Running Sen

Here we discuss steps to take when the errors listed in section 5.4 are encountered and possible permanent fixes. Very briefly:

- if the error is the same as in 5.4.1 there is no need to do anything.
- If any other error Sen needs to be restarted.

To fix the error in 5.4.1 the joint in question needs to be removed from the URDF. As for the segmentation fault in 5.4.2 it is still unclear what caused it so no fixes are suggested. As for the other errors no fixing is required except preventing the UR5 from colliding harmfully. To

prevent the situation in figure 11 the `ur_positioner_kinetic` package needs to be updated to include the ground as a collision surface in its motion planning.

# Chapter 7

## Conclusion

This project set out to answer the question:

In learning affordances, does tactile data combined with visual data improve the performance of a robotic system in detecting tool dependent affordances?

In answering this question, the main constraint faced was the need for a simulation environment setup to test data gathering software that was needed to gather the necessary data useful for answering the given question.

This report documents the key steps taken to develop such a simulation environment setup which started from making the choice of what software and hardware components and tools were used, then went onto simulating models of the hardware components in the simulation environment, to interfacing with these simulated models, on to creating methods for capturing data within the simulation environment, and finally implementing automatic data storage methods to simplify the process of capturing data from within the simulation environment.

The key challenges faced were:

- A significant learning curve involved in first time use software tools such as ROS and Gazebo.
- Outdated and sometimes incomplete documentation for open source libraries such as the Moveit! Library.
- An open source community whose work had not caught up to date with the versions of software that this project used.

The key deliverable achieved was the delivery of a simulation environment setup, Sen, which has the following features:

- an interface to control a simulated robotic arm and multi-digit anthropomorphic robotic hand.
- An interface to a simulated RGB-D camera through which images taken from within the simulation environment can be saved.
- A simple interface to interact with Convolutional Neural Networks.
- A simple method for viewing error messages.

Overall the project's keystone deliverable was achieved and in the process an important tool for future work geared towards answering the question this project set out to investigate was contributed to QMUL research community. The benefits of improving the capabilities of assistive robots to detect affordances in their environment such as dramatic improvements in safety and productivity as well as major cost reductions make it crucially important that questions such as that posed in this project must be answered if assistive robots are to deliver the promise of a better life for all.

# Chapter 8

## Further Work

There is further work to be done in completing Sen's features such as:

- Adding a simulated robotic hand end-effector to the UR5 model.
- Adding tactile sensors to the simulation of the robotic hand
- Creating an interface to access and store this tactile data

There is also work to be done in fixing the segmentation fault error described in 5.4.2, as well as the shoulder\_pan joint error described in 5.4.1. Also, the size of the existing code base can be reduced by deleting code which is not used. The installation procedure can be simplified by using a setup script, and by removing dependencies of libraries such as moveit-visual-tools which are not used.

After these Sen would be fully ready for testing data gathering software, and so future work at that stage would be geared towards gathering data and training a CNN to detect affordances, in order to answer the main question posed by this project.

# Appendix A

## Readme

A simulation setup in gazebo called Sen, equipped for testing data-gathering software which are intended for use on a UR5 arm affixed with an Allegro hand end-effector.

### Features

An interface to control a UR5 arm.

An interface to a simulated RGB-D camera through which images taken from within the simulation environment can be saved.

Planned: An interface to simulated tactile sensors through which contact and force measurements taken from within the simulation environment can be saved.

A simple interface to interact with Convolutional Neural Networks.

## Setting Up Your PC

I highly recommend using a native install of Ubuntu but [Virtualbox](#) can also be used for getting started quickly, however note that image streaming from Gazebo in a VM does not work, and there is a consistent error with the universal\_robot ros package when it is run in a VM. The error is: ERROR: cannot launch node of type [controller\_manager/controller\_manager]: controller\_manager. Also, if using Virtualbox be sure to install the guest additions and extension packs for your Virtualbox if they are required.

A VM will be very problematic but if it is the only option you can download [this ova](#). The password is simply password. It currently has the following installed:

- [Ubuntu 16.04 64-bit PC\(AMD64\)](#)



- [Ros Kinetic](#)
- Git
- Terminator
- [Catkin command line tools](#) - for using catkin build

## Clone This Repository

git clone -b Nnadozie\_ <https://github.com/samisnotinsane/arq-teleop-robot.git>

## Need To Have

- Everything installed in the OVA listed above, except terminator. See [this guide](#) for instructions on installing them.
- [Virtualenv](#) \*helpful for isolating python environments when working with tensorflow. I advice doing all work and installations in this environment.
- [Tensorflow](#) \*note that Tensorflow requires 64-bit architectures.
- After installing tensorflow in virtualenv make sure you are NOT in virtualenv then install moveit!:
- [moveit!](#)

## Starting Up Your Environment

Open a new terminal

### Building the required packages

- The following must be built and sourced in the order given:
- from arq-teleop-robot run cd cam\_ws/
- run catkin\_make
- run source devel/setup.bash
- from arq-teleop-robot run cd ur\_workspace/
- run catkin\_make
- run source devel/setup.bash

- from arq-teleop-robot cd catkin\_ws/
- run sudo apt-get install ros-kinetic-moveit-visual-tools
- run catkin build ur\_positioner\_kinetic
- run source devel/setup.bash

## 0. To Source Built Packages Each Time a New Terminal is Opened

Be sure to change to the actual path

- echo "source <path to cam\_ws>/devel/setup.bash" >> ~/.bashrc
- echo "source <path to ur\_workspace>/devel/setup.bash" >> ~/.bashrc
- echo "source <path to catkin\_ws>/devel/setup.bash" >> ~/.bashrc

## Running Sen

Open a new terminal so your environments are sourced

### 1. Just Sen Publishing Images to a Topic

- If you ignored steps in 0, source the cam\_ws and ur\_workspace environments source <path to cam\_ws>/devel/setup.bash source <path to ur\_workspace>/devel/setup.bash
- Run roslaunch rrobot\_gazebo rrobot\_world.launch
- to view images being published in a new tab run rqt\_image\_view
- select /rrbot/camera1/image\_raw when the viewing terminal opens
- The camera robot can be moved around in Gazebo for different points of view

### 2. Using Sen's UR5 Keyboard Interface

- Run steps in 1.
- In a new terminal make sure the catkin\_ws and ur\_workspace environments are sourced source <path to catkin\_ws>/devel/setup.bash source <path to ur\_workspace>/devel/setup.bash

- Run `roslaunch ur5_moveit_config ur5_moveit_planning_execution.launch sim:=true limited:=true`
- In a new terminal launch the `ur_positioner_kinetic` node `roslaunch ur_positioner_kinetic ur_positioner_kinetic.launch`
- In yet another terminal run `python <path to arq-teleop-robot>/catkin_ws/src/ur_keyboard_controller.py`
- If there is a collision with the ground plane start again from step 1, otherwise the UR5 arm should have moved into a starting position shown in figure

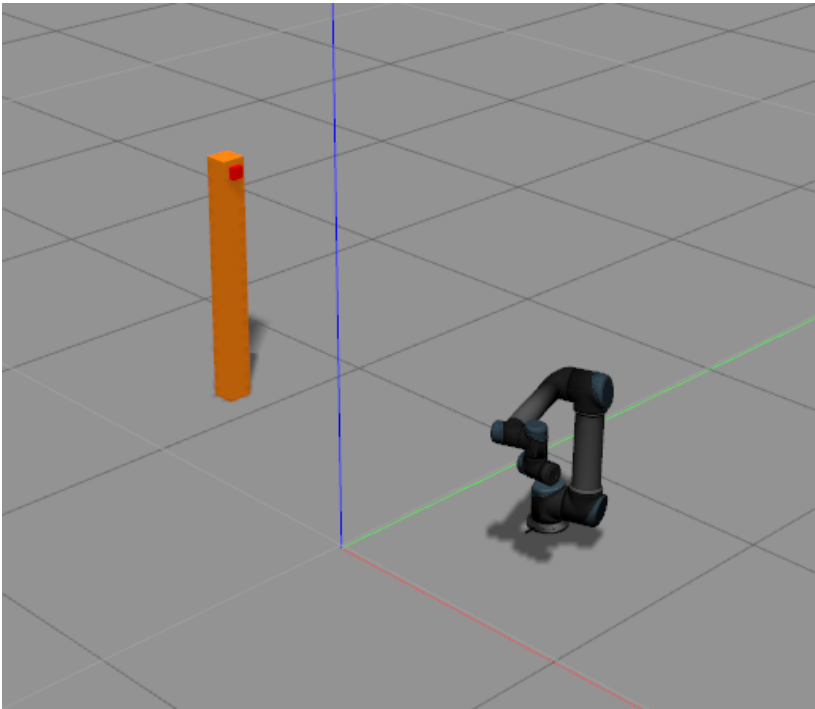


Figure 18: The UR5 arm in the starting position for keyboard control

## Keyboard usage

u for up on the blue axis

j for down on the blue axis

w for up on the red axis

s for down on the red axis

a for left on the green axis

d for right on the green axis

x to exit

### 3. Automatically Saving Sen's Images and Classifying them

- Run steps in 1.
- Run `mkdir <path to arq-teleop-robot>/cam_ws/src/<images>`
- Open the file called `classify.sh` in `arq-teleop-robot/cam_ws/images/`
- Modify the absolute path `/home/lorenzo/doz` in the argument to `--image_file`, to reflect your computers absolute path
- navigate to `/cam_ws/src`
- Run `source <your_path_to_tensorflow>/bin/activate`
- You should see `(tensorflow) qmul@qmul-VirtualBox:~$` in place of just `qmul@qmul-VirtualBox:~$`
- Run `python classify_gazebo_image.py`
- The classification output is stored in `/cam_ws/src/classification.log`
- The first time it runs expect to see the logs for Downloading inception. I suggest renaming the images folder, creating a new images folder and renaming the `classification.log` file each time a session is completed.

# Appendix B

## Risk Assessment

This appendix contains the risk register and accompanying material used in evaluating the risks associated with this project. It is useful for risk management and comprises information pertaining to a four-step risk management process that involves identifying risks, analyzing risks, addressing risks, and monitoring risks. When analyzing risks, the probability of each risk occurring is considered. When addressing risks, the concern is to avoid the risk or reduce the impact of an occurred risk. When monitoring risks, the purpose is to revise risk probabilities and mitigations according to any information that comes to light. The risks considered throughout this project are by no means exhaustive and can expect to vary with differing work situations.

### A.1 Risks in Academic and Commercial Projects

This section provides a non-exhaustive table, table 1, of common risks associated with academic projects and in table 2 common risks associated with commercial projects. It is used when going through the risk management process, for a broad consideration of possible risks as each risk, if irrelevant, can aid in thinking up similar but relevant risks. This section also contains information about two risk assessment techniques: Event tree analysis and Fault tree analysis.

Table 1: Common risks associated with academic projects

Risk
Failure to access required information

-	Non-availability of hardware components or computer resources
-	Hardware or software not suitable for purpose
-	Failure to debug software or correctly assemble hardware
-	Human participants not found
	Testing inadequate
	Poor time management
	Electric shock
	Burning when soldering
-	Injury when using tools to construct enclosures or cut circuit boards
-	Injury caused by out of control robot

Table 2: Common risks associated with commercial projects

	Risk
	Experienced staff leave project
-	New management in organization has different priorities
-	Essential hardware or equipment not delivered on time
-	Unanticipated changes to requirement

Essential interface specifications behind  
 schedule  
 System larger than expected  
  
 Underlying technology for system is  
 superseded  
 Competitor is faster to market similar product

### A.1.1 Event Tree Analysis

This section contains a description of event tree analysis, together with an example.. Event tree analysis proceeds from an accidental event, defined as the first significant deviation from a normal situation that may lead to unwanted consequences. Examples are a burst pipe or gas explosion. Such accidents may lead to numerous outcomes, which can be illustrated by a consequence spectrum. Consider the event of a fire in a building with a sprinkler system. Figure 19 depicts event tree analysis of this accident, illustrated using a consequence spectrum.

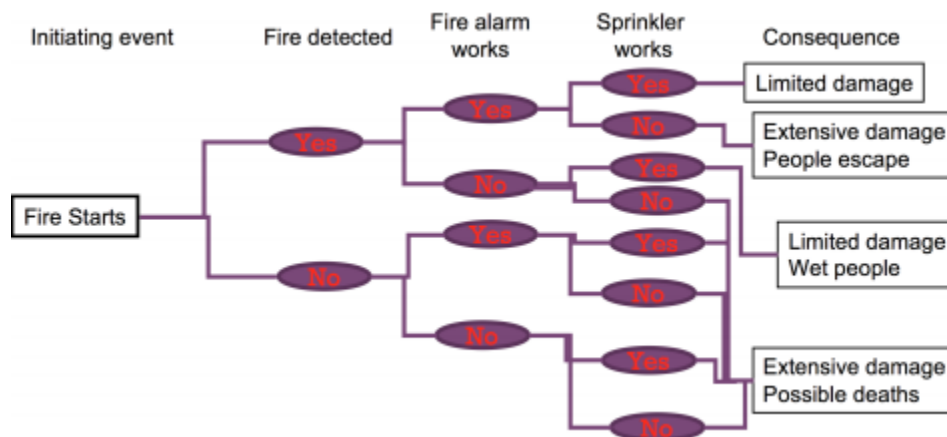


Figure 19: Event tree analysis of a fire accident in a building with sprinkler system, showing different consequences illustrated using a consequence spectrum (61)

## A.1.2 Fault Tree Analysis

This section contains a description of fault tree analysis, together with an example. Fault tree analysis is mainly employed for safety-critical systems. An undesired effect is considered the top event, from which all possible paths leading to it are represented using logic gate symbols. Probability values may then be assigned to each lower-level event, and the probability of the top event occurring can be calculated by identifying the shortest path. Consider the event of a brake failure. Figure 20 depicts fault tree analysis of this event.

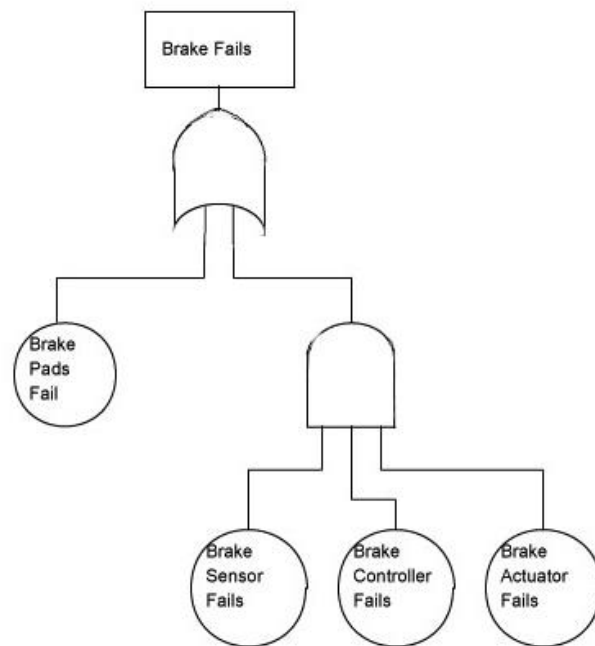


Figure 20: Fault tree analysis of a brake failure event (62)



## A.2 Risk Register

This section identifies the risks to successful completion of this project in table 3, and any risks to personnel. These risks may be broadly categorized as project risks, product risks, or business risks. Project risks are those that affect project schedules or resources. Product risks affect quality or performance of a product. While business risks affect the organization or individual developing a product.

This section is supplemented by a risk likelihood scale (table 4) and risk impact scale (table 5).

Description of risk	Description of impact	Likelihood rating	Impact rating	Preventive actions	Post-risk actions
Cannot generate or find adequate tactile data for training Cen	Will result in a practically undelivered project	4	4	Seek more man-power to generate training data	Complete project using the training data available
Cannot find adequate man-power to generate training data	Will result in a practically undelivered project	3	4	Provide incentives such as payment to secure adequate man-power	Complete project using the training data available
Tactile sensors not ready in time for real world testing	Will result in a mostly complete project	4	3	Help course-mate in preparing custom built tactile sensors	Complete project using only simulation environment and disregard

Unable to create simulations needed for experimentation and training	Will completely prevent project from being delivered	3	5	Seek out experienced individuals to provide help in doing this	real world demonstration Complete project using readily available simulation materials which exist although they differ from the available physical hardware
Unable to build back end neural network	Will completely prevent project from being delivered	3	5	Pre-emptively find someone with expertise in doing this who will be willing to provide guidance	Deliver what can be delivered without a neural network such as a simulation environment and control interface
Unable to find knowledgeable experts that can help me solve esoteric problems I have difficulty with	Will result in a mostly complete project	3	3	Develop expertise in the areas which are required through personal study	Deliver can be delivered even though it is not the desired standard

Unable to find documentation for open source software which is needed for the project	Will result in a practically undelivered project	3	4	Use open source tools that have ample documentation and a strong community	Seek help from contributors to these open source projects
Physical injury to lab personnel caused by equipment	Impact is very hard to predict	2	1 - 5	Follow safety procedures	Seek medical help for the affected personnel, revise safety procedures, and deliver what can be delivered

Table 3: Analysis of project associated risks detailing occurrence-minimization actions and post-occurrence impact reduction actions

Weighting	Description
1	Will not happen
2	Very unlikely to happen
3	Equally likely and unlikely to happen
4	Very likely to happen

5	Will happen
---	-------------

Table 4: Likelihood scale

Weighting	Description
1	Will not impact project delivery
2	Effect to project delivery can be ignored
3	Will result in a mostly complete project
4	Will result in a practically undelivered project
5	Will completely prevent project from being delivered

Table 5: Impact scale

# References

1. Jamone L, Ugur E, Cangelosi A, Fadiga L, Bernardino A, Piater J et al. Affordances in Psychology, Neuroscience, and Robotics: A Survey. *IEEE Transactions on Cognitive and Developmental Systems*. 2018;10(1):4-25.
2. Horton T, Chakraborty A, Amant R. Affordances for robots: a brief survey. *Avant: Journal of Philosophical-Interdisciplinary Vanguard*. 2012;3(2/2012):70-84.
3. Affordances and Emergence of Concepts. *Proceedings of the Tenth International Conference on Epigenetic Robotics*. 2010;:11 – 18.
4. Griffith S, Sinapov J, Sukhoy V, Stoytchev A. A Behavior-Grounded Approach to Forming Object Categories: Separating Containers From Noncontainers. *IEEE Transactions on Autonomous Mental Development*. 2012;4(1):54-69.
5. KUNIYOSHI Y, FUKANO R, OTANI T, KOBAYASHI T, OTSU N. HAPTIC DETECTION OF OBJECT AFFORDANCES BY A MULTI-FINGERED ROBOT HAND. *International Journal of Humanoid Robotics*. 2005;02(04):415-435.
6. Fitzpatrick P, Carello C, Schmidt R, Corey D. Haptic and Visual Perception of an Affordance for Upright Posture. *Ecological Psychology*. 1994;6(4):265-287.
7. Klevberg G, Anderson D. Visual and haptic perception of postural affordances in children and adults. *Human Movement Science*. 2002;21(2):169-186.
8. Montesano L, Lopes M, Bernardino A, Santos-Victor J. Learning Object Affordances: From Sensory--Motor Coordination to Imitation. *IEEE Transactions on Robotics*. 2008;24(1):15-26.
9. Ridge B, Skocaj D, Leonardis A. Self-supervised cross-modal online learning of basic object affordances for developmental robotic systems. 2010 *IEEE International Conference on Robotics and Automation*. 2010;.
10. Baleia J, Santana P, Barata J. On Exploiting Haptic Cues for Self-Supervised Learning of Depth-Based Robot Navigation Affordances. *Journal of Intelligent & Robotic Systems*. 2015;80(3-4):455-474.

11. Nguyen A, Kanoulas D, Caldwell D, Tsagarakis N. Detecting object affordances with Convolutional Neural Networks. 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2016;.
12. Tikhonoff V, Pattacini U, Natale L, Metta G. Exploring affordances and tool use on the iCub. 2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids). 2013;.
13. Myers A, Teo C, Fermuller C, Aloimonos Y. Affordance detection of tool parts from geometric features. 2015 IEEE International Conference on Robotics and Automation (ICRA). 2015;.
14. Michaels C, Weier Z, Harrison S. Using Vision and Dynamic Touch to Perceive the Affordances of Tools. *Perception*. 2007;36(5):750-772.
15. A Guide to TF Layers: Building a Convolutional Neural Network | TensorFlow [Internet]. TensorFlow. 2018 [cited 24 April 2018]. Available from: [https://www.tensorflow.org/tutorials/layers#intro\\_to\\_convolutional\\_neural\\_networks](https://www.tensorflow.org/tutorials/layers#intro_to_convolutional_neural_networks)
16. CS231n Convolutional Neural Networks for Visual Recognition [Internet]. Cs231n.github.io. 2018 [cited 24 April 2018]. Available from: <https://cs231n.github.io/convolutional-networks/>
17. Image classification with Deep Learning, CNN, Caffe, OpenCV 3.x and CUDA - Robotic Vision | Machine Learning | Software Development [Internet]. Robotic Vision | Machine Learning | Software Development. 2018 [cited 24 April 2018]. Available from: <http://www.coldvision.io/2016/07/29/image-classification-deep-learning-cnn-caffe-opencv-3-x-cuda/>
18. Robot Kinematics [Internet]. Southampton.ac.uk. 2018 [cited 24 April 2018]. Available from: <http://www.southampton.ac.uk/~rmc1/robotics/arkinematics.htm>
19. Inverse kinematics [Internet]. En.wikipedia.org. 2018 [cited 24 April 2018]. Available from: [https://en.wikipedia.org/wiki/Inverse\\_kinematics](https://en.wikipedia.org/wiki/Inverse_kinematics)
20. Mathematics for Inverse Kinematics [Internet]. Cs.cmu.edu. 2018 [cited 24 April 2018]. Available from: <http://www.cs.cmu.edu/~15464-s13/lectures/lecture6/IK.pdf>
21. Jacobian methods for inverse kinematics and planning [Internet]. Homes.cs.washington.edu. 2018 [cited 24 April 2018]. Available from: [https://homes.cs.washington.edu/~todorov/courses/cseP590/06\\_JacobianMethods.pdf](https://homes.cs.washington.edu/~todorov/courses/cseP590/06_JacobianMethods.pdf)
22. Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods [Internet]. Graphics.cs.cmu.edu. 2018 [cited 24 April

- 2018]. Available from: <http://graphics.cs.cmu.edu/nsp/course/15464-s17/lectures/iksurvey.pdf>
23. Treiber M. An introduction to object recognition. London [u.a.]: Springer; 2013.
  24. Documentation - Point Cloud Library (PCL) [Internet]. Pointclouds.org. 2018 [cited 25 April 2018]. Available from: <http://pointclouds.org/documentation/tutorials/#recognition-tutorial>
  25. OpenCV: OpenCV Tutorials [Internet]. Docs.opencv.org. 2018 [cited 25 April 2018]. Available from: [https://docs.opencv.org/3.4.1/d9/df8/tutorial\\_root.html](https://docs.opencv.org/3.4.1/d9/df8/tutorial_root.html)
  26. Mallick S. Image Recognition and Object Detection : Part 1 | Learn OpenCV [Internet]. Learnopencv.com. 2018 [cited 25 April 2018]. Available from: <https://www.learnopencv.com/image-recognition-and-object-detection-part1/>
  27. Khoshelham K, Elberink S. Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications. Sensors. 2012;12(2):1437-1454.
  28. Rusu R, Cousins S. 3D is here: Point Cloud Library (PCL). 2011 IEEE International Conference on Robotics and Automation. 2011;.
  29. Point Cloud Library (PCL): Module segmentation [Internet]. Docs.pointclouds.org. 2018 [cited 25 April 2018]. Available from: [http://docs.pointclouds.org/trunk/group\\_\\_segmentation.html](http://docs.pointclouds.org/trunk/group__segmentation.html)
  30. Documentation - Point Cloud Library (PCL) [Internet]. Pointclouds.org. 2018 [cited 25 April 2018]. Available from: <http://pointclouds.org/documentation/tutorials/#segmentation-tutorial>
  31. Point Cloud Library (PCL): Module sample\_consensus [Internet]. Docs.pointclouds.org. 2018 [cited 25 April 2018]. Available from: [http://docs.pointclouds.org/trunk/group\\_sample\\_consensus.html](http://docs.pointclouds.org/trunk/group_sample_consensus.html)
  32. Model Free 3D Object Detection & tracking PCL - ROS Answers: Open Source Q&A Forum [Internet]. Answers.ros.org. 2018 [cited 25 April 2018]. Available from: <https://answers.ros.org/question/244797/model-free-3d-object-detection-tracking-pcl/>
  33. Jianxin Wu, Reh J. CENTRIST: A Visual Descriptor for Scene Categorization. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2011;33(8):1489-1501.
  34. Introduction to SIFT (Scale-Invariant Feature Transform) — OpenCV 3.0.0-dev documentation [Internet]. Docs.opencv.org. 2018 [cited 25 April 2018]. Available from: [https://docs.opencv.org/3.0-beta/doc/py\\_tutorials/py\\_feature2d/py\\_sift\\_intro/py\\_sift\\_intro.html#goal](https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html#goal)

35. Lowe D. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision. 2004;60(2):91-110.
36. Spatial envelope [Internet]. People.csail.mit.edu. 2018 [cited 25 April 2018]. Available from: <http://people.csail.mit.edu/torralba/code/spatialenvelope/>
37. ROS.org | Powering the world's robots [Internet]. Ros.org. 2018 [cited 25 April 2018]. Available from: <http://www.ros.org/>
38. Gazebo [Internet]. GazeboSim.org. 2018 [cited 25 April 2018]. Available from: <http://gazeboSim.org/>
39. rviz - ROS Wiki [Internet]. Wiki.ros.org. 2018 [cited 25 April 2018]. Available from: <http://wiki.ros.org/rviz>
40. Robots/AllegroHand - ROS Wiki [Internet]. Wiki.ros.org. 2018 [cited 25 April 2018]. Available from: <http://wiki.ros.org/Robots/AllegroHand#>
41. TensorFlow [Internet]. TensorFlow. 2018 [cited 25 April 2018]. Available from: <https://www.tensorflow.org/>
42. universal\_robot - ROS Wiki [Internet]. Wiki.ros.org. 2018 [cited 25 April 2018]. Available from: [http://wiki.ros.org/universal\\_robot](http://wiki.ros.org/universal_robot)
43. MoveIt! Motion Planning Framework [Internet]. MoveIt.ros.org. 2018 [cited 25 April 2018]. Available from: <https://moveit.ros.org/>
44. Kinect Sensor [Internet]. Msdn.microsoft.com. 2018 [cited 25 April 2018]. Available from: <https://msdn.microsoft.com/en-gb/library/hh438998.aspx>
45. Gazebo : Tutorial : Which combination of ROS/Gazebo versions to use [Internet]. GazeboSim.org. 2018 [cited 25 April 2018]. Available from: [http://gazeboSim.org/tutorials?tut=ros\\_wrapper\\_versions](http://gazeboSim.org/tutorials?tut=ros_wrapper_versions)
46. felixduvallet/allegro-hand-ros [Internet]. GitHub. 2018 [cited 25 April 2018]. Available from: <https://github.com/felixduvallet/allegro-hand-ros>
47. simlabrobotics/allegro\_hand\_ros [Internet]. GitHub. 2018 [cited 25 April 2018]. Available from: [https://github.com/simlabrobotics/allegro\\_hand\\_ros](https://github.com/simlabrobotics/allegro_hand_ros)
48. Gazebo : Tutorial : URDF in Gazebo [Internet]. GazeboSim.org. 2018 [cited 25 April 2018]. Available from: [http://gazeboSim.org/tutorials/?tut=ros\\_urdf](http://gazeboSim.org/tutorials/?tut=ros_urdf)
49. Revision history - Gazebo: Q&A Forum [Internet]. Answers.gazeboSim.org. 2018 [cited 25 April 2018]. Available from: <http://answers.gazeboSim.org/questions/17457/revisions/>
50. Topics - ROS Wiki [Internet]. Wiki.ros.org. 2018 [cited 25 April 2018]. Available from: <http://wiki.ros.org/Topics>



51. GitHub [Internet]. En.wikipedia.org. 2018 [cited 25 April 2018]. Available from:  
<https://en.wikipedia.org/wiki/GitHub>
52. shunchan0677/Tensorflow\_in\_ROS [Internet]. GitHub. 2018 [cited 25 April 2018].  
Available from: [https://github.com/shunchan0677/Tensorflow\\_in\\_ROS](https://github.com/shunchan0677/Tensorflow_in_ROS)
53. Move Group Interface Tutorial — moveit\_tutorials Kinetic documentation [Internet].  
Docs.ros.org. 2018 [cited 25 April 2018]. Available from:  
[http://docs.ros.org/kinetic/api/moveit\\_tutorials/html/doc/pr2\\_tutorials/planning/src/doc/move\\_group\\_interface\\_tutorial.html](http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/pr2_tutorials/planning/src/doc/move_group_interface_tutorial.html)
54. Programming for Robotics Introduction to ROS [Internet]. Ethz.ch. 2018 [cited 25 April 2018]. Available from: <https://www.ethz.ch/content/dam/ethz/special-interest/mavt/robotics-n-intelligent-systems/rsl-dam/ROS2017/lecture2.pdf>
55. How to get image without storing into disk from gazebo ? - Gazebo: Q&A Forum [Internet]. Answers.gazebosim.org. 2018 [cited 25 April 2018]. Available from:  
<http://answers.gazebosim.org/question/16214/how-to-get-image-without-storing-into-disk-from-gazebo/?answer=16977#post-id-16977>
56. osrf / gazebo / source / examples / plugins / camera — Bitbucket [Internet].  
Bitbucket.org. 2018 [cited 25 April 2018]. Available from:  
<https://bitbucket.org/osrf/gazebo/src/651b6b1fc78b05ae2825d3771371f6824f83e579/examples/plugins/camera/>
57. Gazebo : Tutorial : Gazebo plugins in ROS [Internet]. Gazebosim.org. 2018 [cited 25 April 2018]. Available from: [http://gazebosim.org/tutorials?tut=ros\\_gzplugins](http://gazebosim.org/tutorials?tut=ros_gzplugins)
58. Gazebo : Tutorial : Force/Torque Sensor [Internet]. Gazebosim.org. 2018 [cited 25 April 2018]. Available from:  
[http://gazebosim.org/tutorials?tut=force\\_torque\\_sensor&cat=sensors](http://gazebosim.org/tutorials?tut=force_torque_sensor&cat=sensors)
59. Gazebo : Tutorial : Contact Sensor [Internet]. Gazebosim.org. 2018 [cited 25 April 2018]. Available from: [http://gazebosim.org/tutorials?tut=contact\\_sensor](http://gazebosim.org/tutorials?tut=contact_sensor)
60. Image Recognition | TensorFlow [Internet]. TensorFlow. 2018 [cited 26 April 2018].  
Available from: [https://www.tensorflow.org/tutorials/image\\_recognition](https://www.tensorflow.org/tutorials/image_recognition)
61. EECS UG Project - 2017/18 [Internet]. Qmplus.qmul.ac.uk. 2018 [cited 26 April 2018].  
Available from:  
[https://qmplus.qmul.ac.uk/pluginfile.php/964285/mod\\_resource/content/5/5-Interim%20Report%20and%20Risk%20Assessment%20%20slides%20per%20page.pdf](https://qmplus.qmul.ac.uk/pluginfile.php/964285/mod_resource/content/5/5-Interim%20Report%20and%20Risk%20Assessment%20%20slides%20per%20page.pdf)

62. Safety Critical Systems Analysis [Internet]. Users.ece.cmu.edu. 2018 [cited 26 April 2018]. Available from:  
[http://users.ece.cmu.edu/~koopman/DES\\_s99/safety\\_critical/index.html](http://users.ece.cmu.edu/~koopman/DES_s99/safety_critical/index.html)
63. Hall F. The Astronomical Journal [Internet]. 2018 [cited 26 April 2018]. Available from:  
[https://www.universal-robots.com/media/50588/ur5\\_en.pdf](https://www.universal-robots.com/media/50588/ur5_en.pdf)
64. Allegro Hand is a low-cost and highly adaptive robotic hand. [Internet]. Simlab.co.kr. 2018 [cited 26 April 2018]. Available from: <http://www.simlab.co.kr/Allegro-Hand.htm#Specifications>
65. Kinect for Windows Sensor Components and Specifications [Internet]. Msdn.microsoft.com. 2018 [cited 26 April 2018]. Available from:  
<https://msdn.microsoft.com/en-us/library/jj131033.aspx>